$8.00

A PAPERBYTE™ BOOK

# LINK68

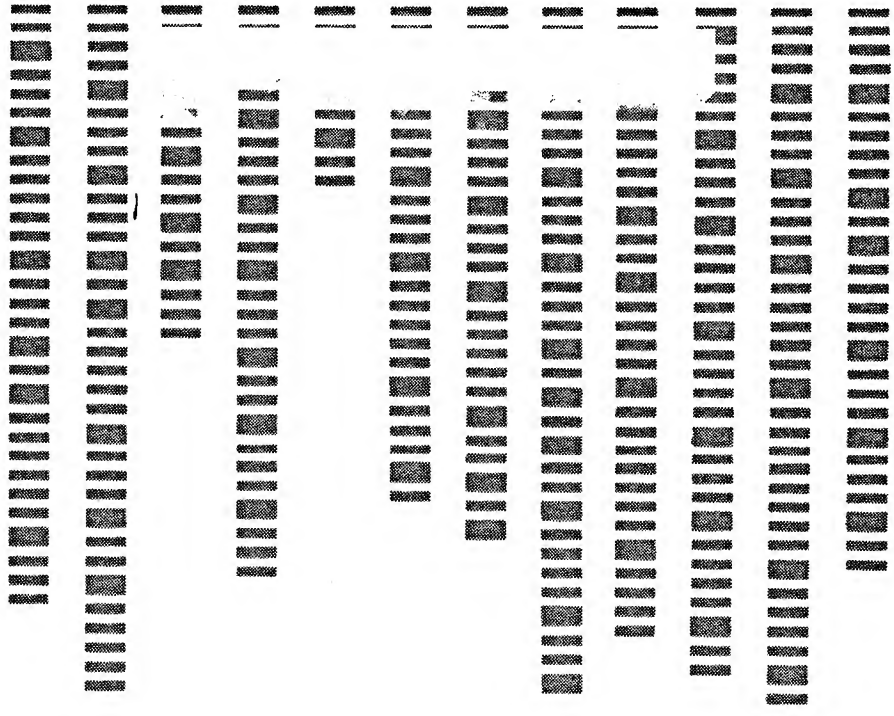
## AN M6800 LINKING LOADER

by Robert D. Cramer and Jack E. Hemenway

# LINK6800

## AN M6800 LINKING LOADER

by Robert D. Grappel and Jack E. Hemenway



BYTE Publications, Inc.
70 Main Street
Peterborough, New Hampshire 03458

The authors of the programs provided with this book have carefully reviewed them to ensure their performance in accordance with the specifications described in the book. The authors, however, make no warranties whatever concerning the programs, and assume no responsibility or liability of any kind for errors in the programs or for the consequences of any such errors. The programs are the sole property of the authors and have been registered with the United States Copyright Office.



**Library of Congress Cataloging in Publication Data**

Grappel, Robert D.
LINK68-linking loader for the Motorola M6800.

Includes index.
1. LINK68 (Computer program) 2. Linking loaders (Computer programs) 3. Motorola 6800 (Computer) – Programming. I. Hemenway, Jack E., joint author. II. Title.
QA76.8.M67G72 001.6'425 78-17819
ISBN 0-931718-09-0

Printed in the United States of America

# Table of Contents

# To Begin With . . .

LINK68 is a one pass linking loader used to load and link object files produced by the Resident 6800 Macro Assembler RA6800ML (available as a PAPERBYTE™ book). It allows separately translated relocatable object modules to be loaded and linked together to form a single executable load module. LINK68 produces a Load Map and a load module in Motorola MIKBUG loader format.

The Linking Loader requires 2 K bytes of memory, a system console such as a Teletype, a system monitor such as the Motorola MIKBUG read only memory program of the ICOM Floppy Disk Operating System (FDOS), and some form of mass file storage such as dual cassette recorders of a floppy disk. A system monitor other than those mentioned above could be used by changing two IO jumps in the Linking Loader (a jump to the terminal character input routine INEEE and a jump to the terminal character output routine OUTEEE) and by supplying functionally equivalent IO routines for the user's specific system.

This book is divided into three major sections. THE LINKING LOADER provides detailed descriptions of the major routines of the Linking Loader. Included are details about the various routine linkages, pointers, flags, etc. This section provides the necessary background for using the Linking Loader as well as understanding its basic operations.

INTERFACING AND USING THE LINKING LOADER gives information about the IO conventions used, execution of the Linking Loader, and the input file format. Naturally, the exact IO interface needed for using the Linking Loader depends on the actual configuration of the user's system. Therefore, tips are given on how to design IO routines or modify those provided as examples to fit the user's system. This section concludes with a discussion of the methods of preparing the loader for routine use in your system by reading it in and relocating it with a "bootstrap" version pre-linked in absolute format.

The third section is the set of appendices which contain error messages generated by the Linking Loader, the Linking Loader and sample IO driver assembly listings, the bar code representations of the various relocatable object modules of the loader, and an implementation guide for bootstrapping LINK68 and linking loader and IO routines in absolute formats for the bootstrap process.

Finally, a detailed INDEX is included for quick cross-reference to the Linking Loader's routines.

In this book is what we believe to be a complete set of documentation for the Linking Loader. Every flowchart, every listing, every item was included for one purpose: to provide the user with everything needed for the use or modification of the Linking Loader.

In addition, it was the express purpose of the authors to provide everything necessary so that the user can easily learn what he or she needs to know about the system. By providing not only the source code and bar code listings, but also a detailed description of the major routines of the Linking Loader, we intend to provide the user with an opportunity to learn about the nature of linking loader design and implementation, as well as simply acquiring a useful software tool. It is through this kind of encouragement that we hope to advance the state of the art of home computing.

*Robert D. Grappel*
*Jack E. Hemenway*

# The Linking Loader

LINK68 is a one pass Linking Loader designed to load and link relocatable input modules that were separately prepared by assembly or compilation.

The input to LINK68 is the set of output modules produced by the Resident 6800 Macro Assembler RA6800ML (available as a PAPERBYTE™ book). LINK68 links the modules by matching Entry symbols in one module with External symbols in another module(s); it relocates the modules by assigning absolute addresses to relocatable address fields; it assigns absolute addresses from the Common area to address fields which refer to the Common area; and finally, it prints a Load Map which displays the Entry symbols and their assigned absolute addresses.

As LINK68 reads the input object modules, it keeps track of the Entry symbol definitions and External symbol references by storing them in a Symbol Table. The External references are matched with the Entry definitions and the correct absolute address of the Entry symbol is placed into the External reference's address field, completing the linkage.

LINK68 handles the relocation of address fields marked as relocatable in the input module(s) by adding the starting address of the module being loaded to the offset in the address field marked relocatable. This absolute address is then placed into the address field of the loaded module.

Address fields marked as Common in the input modules are handled by adding the starting address of the Common area to the offset in the address field and placing the sum into the address field of the loaded module.

The Load Map (see figure 1) provides information about the loaded and linked module. The first line gives the starting and ending addresses of the load module. This is followed by a listing of all the Entry points in the loaded module along with their absolute addresses. If any of the Entry points are either unresolved or redefined this information is printed next to the absolute address. An Entry marked as unresolved is an External reference that was not resolved, ie: there was no Entry point found for that External symbol. The last line of the Load Map gives the limits of the Common area.

Following are detailed descriptions of the major routines of the Linking Loader.

## LOAD

This routine is the driving section for the Linking Loader. Bytes from the input object file are read one at a time and, depending on what they are, jumps are made to processing routines to load, relocate, link, etc. When an End of File (EOF) is encountered, the load limits, Load

*Figure 1: Sample Linking Loader run:*

① *: the base address of the new load module.*

② *: the user's upper memory limit.*

③ *: the beginning address of the program's Common area.*

④ *: the beginning and ending addresses of the load module.*

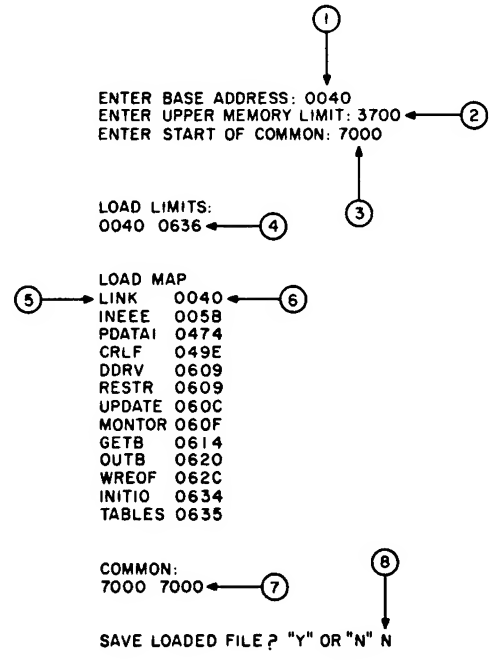
⑤ *: the list of entry symbols for the load module.*

⑥ *: the addresses of the entry symbols for the load module.*

⑦ *: the beginning and ending addresses of the load module's Common area (in this example, the Common area was not used by the program).*

⑧ *: the load module will not be saved.*

*Note that items ① , ② , ③ , and ⑧ are items supplied by the user, and that all remaining items are produced by the Linking Loader.*



Map, and Common limits are printed. Finally the loaded module is saved (if desired) and control returns to the system monitor.

| | |
|---|---|
| Calls: | BADDR, CRLF, GETB, PDATA1 |
| Jumps: | LOADE, LOADM, LOADN, LOADP, LOADR, LOADX |
| Entrys: | LOAD2 |
| Flags: | NFLAG |
| Pointers: | BASE, BASESV, CBAS, CBASSV, HICBAS, LAST, NXTSYM, SYMEND, SYMTAB, TABLE, UPLIM |
| Temporaries: | BYTE |

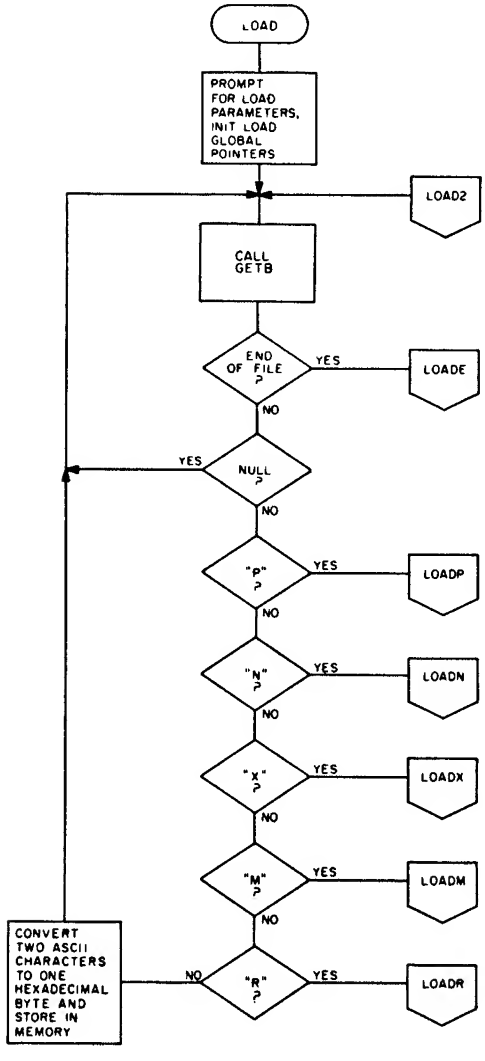



*Figure 2: The flowchart for LOAD, the main Linking Loader routine.*

## LOADE

This routine is executed whenever an End of File (EOF) is encountered in the input object file. The load limits, the Load Map and the Common limits are printed. Next the loader prompts the user to see if the loaded module is to be saved and if so, writes it out in absolute Motorola MIKBUG format. Control is then returned to the system monitor.

| | |
|---|---|
| Called By: | LOAD |
| Calls: | CRLF, INEEE, OUT4HS, PDATA1, PRTSYM |
| Jumps: | MONTOR, PUNCH |
| Pointers: | BASESV, CBAS, HICBAS, LAST |

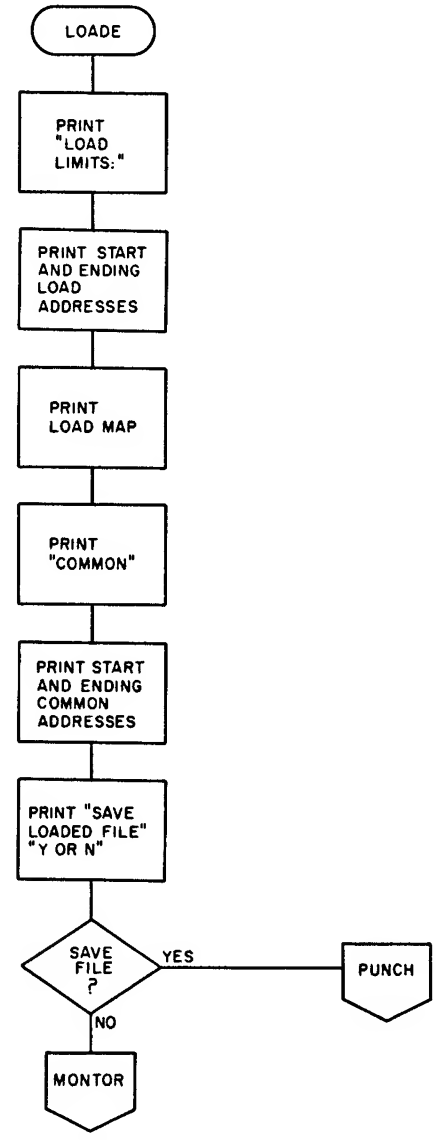



*Figure 3: The flowchart for the LOADE routine.*

## LOADR

This routine relocates an address field in the loaded module by adding the contents of the pointer BASE to the address field. Control is then returned to LOAD2.

Called By: LOAD
Calls: MEMCHK
Jumps: LOAD2
Pointers: BASE

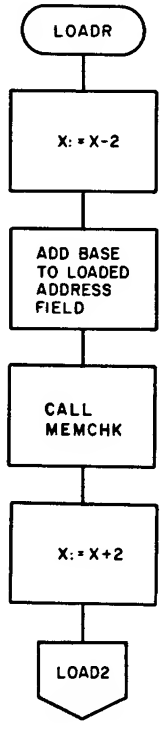



*Figure 4: The flowchart for the LOADR routine.*

## LOADP

This routine is executed whenever a new program is encountered in the input file. The length of Common used by the module is read and saved. The pointer or relocation base is reset to the start of the new module. Control then passes to LOAD2.

Called By: LOAD
Calls: none
Jumps: LOAD2
Pointers: BASE, CBAS, CBASSV, HICBAS

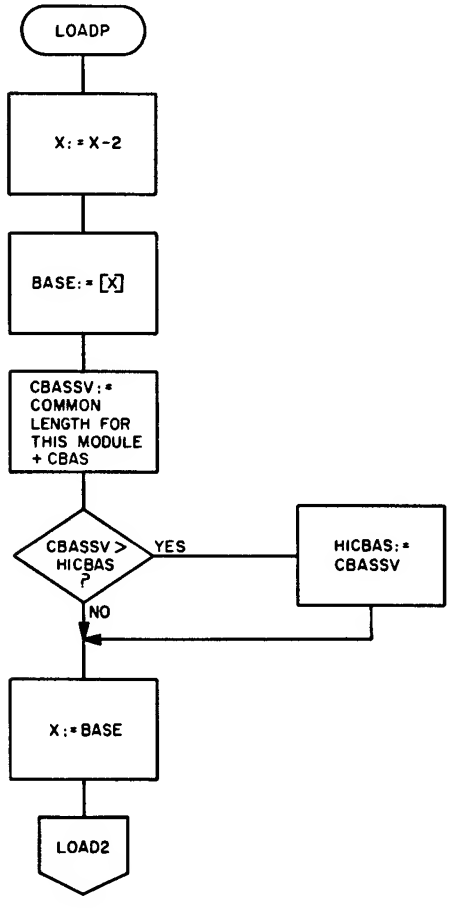



*Figure 5: The flowchart for the LOADP routine.*

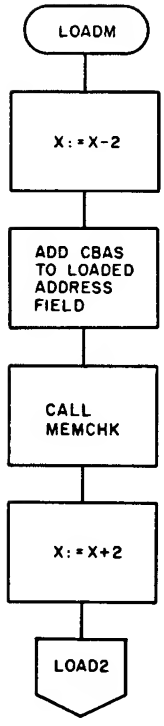
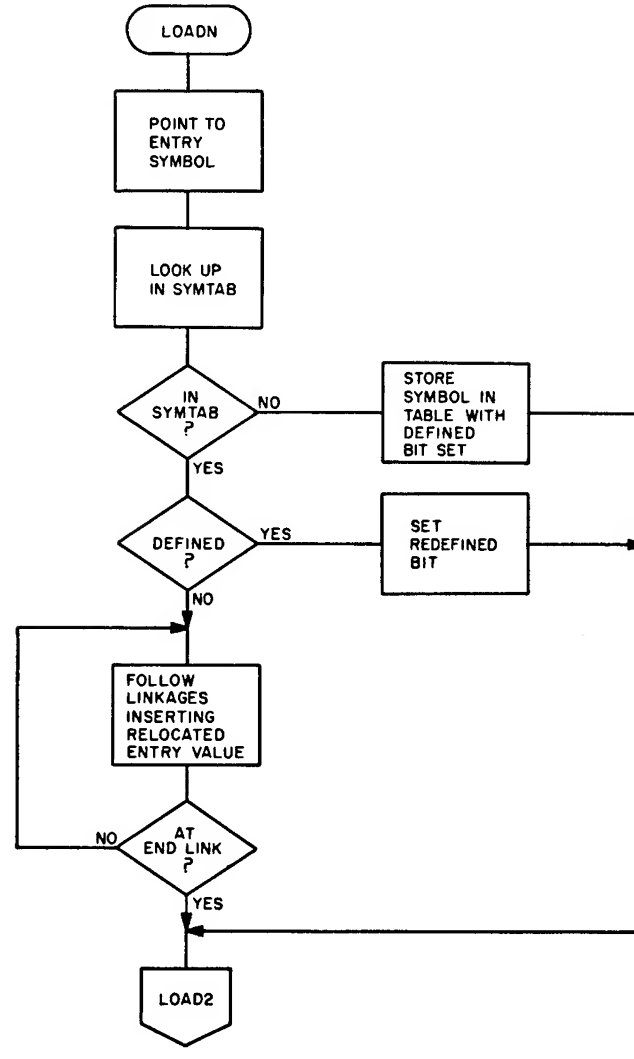



*Figure 6: The flowchart for the LOADM routine.*

*Figure 7: The flowchart for the LOADN routine.*




LOADM

This routine relocates an address field that makes a reference to Common from the loaded module. The contents of CBAS are added to the address field. Control then returns to LOAD2.

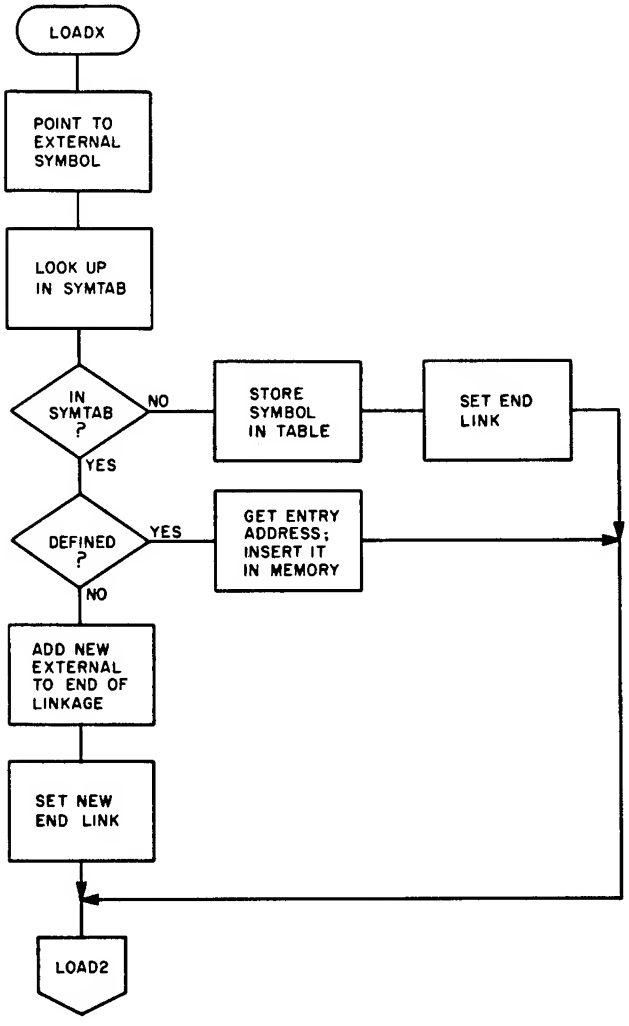
| | |
|---|---|
| Called By: | LOAD |
| Calls: | MEMCHK |
| Jumps: | LOAD2 |
| Pointers: | CBAS |

LOADN

This routine handles the processing of Entry symbols from the input file. When an Entry symbol is found, it and its address value are stored in the Symbol Table (SYMTAB). If the symbol is already in SYMTAB and has had an absolute address already placed in the symbol entry, it is an error. However, symbols may have been stored in SYMTAB by a prior encounter of an External instead of an Entry reference to that symbol. When an External reference is encountered before an Entry reference, the symbol is entered into the Symbol Table without an absolute address entry, and the location of the External reference in the loaded module is linked to the Symbol Table entry using a linked list. LOADN searches this linked list when it finds the symbol is already in SYMTAB and resolves all of the external references linked to the Symbol Table entry.

| | |
|---|---|
| Called By: | LOAD |
| Calls: | LKPSYM, STOSYM |
| Jumps: | LOAD2 |
| Pointers: | DESCRA, DESCRC, LAST, LC, SYMPTR |

*Figure 8: The flowchart for the LOADX routine.*




## LOADX

This routine processes the External symbols found in the input file. It first searches the Symbol Table (SYMTAB) for the External symbol. If it is there and has an associated entry address defined, the address value is taken from the Symbol Table and stored in the address field of the external reference in the loaded module. If it is not already in the Symbol Table, it is stored in the Symbol Table and the external reference location in the loaded module is stored as the symbol address value. The external reference location is set to hexdecimal FFFF, indicating the end of the linked list.

If the External symbol is already in the Symbol Table but there is no Entry address associated with it, then there are one or more External references linked to the Symbol Table entry using a linked list. The new External reference location is then linked onto the end of this list.

| | |
|---|---|
| Called By: | LOAD |
| Calls: | LKPSYM, STOSYM |
| Jumps: | LOAD2 |
| Pointers: | DESCRC, DESCRA, LAST, LC, SYMPTR |

## STOSYM

This routine stores symbols and their address values into the Symbol Table (SYMTAB). A linear table is used and the pointer NXTSYM points to the next available table location. The routine first checks to see if the Symbol Table is full, and if not, stores the symbol at the location pointed to by NXTSYM. It then increments NXTSYM before returning.

| | |
|---|---|
| Called By: | LOADN, LOADX |
| Calls: | PDATA1 |
| Jumps: | MONTOR |
| Pointers: | DESCRA, DESCRC, LC, NXTSYM, SYMEND, SYMPTR, SYMTAB |

## LKPSYM

This routine searches the Symbol Table for a symbol using a linear search. If the symbol is found, SYMPTR points to the location of the symbol in the table, register B is loaded with the INFO-BYTE, and the X register is loaded with the address value of the symbol.

| | |
|---|---|
| Called By: | LOADN, LOADX |
| Calls: | COMPAR |
| Jumps: | none |
| Pointers: | COUNT, DESCRA, DESCRC, NXTSYM, STRNG1, STRNG2, SYMPTR, SYMTAB |

## COMPAR

This routine is used to compare variable length strings. The string lengths can be up to 255 bytes. When COMPAR is called the Index register X points to a parameter list of 5 bytes: bytes 1 and 2 provide the address of the first string; bytes 3 and 4 the address of the second string; and byte 5, the number of bytes to be compared.

On return from this routine, the results of the comparison are reflected in the condition codes register.

Example:

```
BNE NOMATCH   string1 is not equal to string2
BEQ MATCH     string1 is equal to string2
```

| | |
|---|---|
| Called By: | LKPSYM, PRTSYM |
| Calls: | none |
| Flags: | none |
| Pointers: | Index register |
| Temporaries: | XSAV |

## MEMCHK

This routine checks to see if the memory available for the loaded modules is used up. If it is not, the routine returns. If it is, the routine prints an error message and jumps to LOADE to print a partial Load Map.

| | |
|---|---|
| Called By: | LOAD, LOADM, LOADR |
| Calls: | CRLF, PDATA1 |
| Jumps: | LOADE |
| Pointers: | UPLIM |

## PRTSYM

This routine prints the Load Map. It searches the Symbol Table for the lowest valued address entry and then prints the symbol and its address, flagging the entry as printed. The Symbol Table then is searched again, ignoring entries marked as already printed. When all the entries have been printed, the routine returns.

| | |
|---|---|
| **Called By:** | LOADE |
| **Calls:** | COMPAR, CRLF, OUTEEE, OUT4HS, OUTS, PDATA1 |
| **Jumps:** | none |
| **Pointers:** | COUNT, DESCRA, DESCRC, HIVAL, NXTSYM, STRNG1, STRNG2, SYMPTR, SYMTAB |

## PUNCH

This routine outputs the contents of the loaded program in absolute Motorola MIKBUG format. When it has finished, it writes an End of File (EOF) byte, closes the output file, and passes control to the system monitor.

| | |
|---|---|
| **Called By:** | LOADE |
| **Calls:** | OUTB, WREOF |
| **Jumps:** | UPDATE |
| **Pointers:** | BASESV, LAST, MCONT, TEMP |

# Interfacing and Using the Linking Loader

## IO Interface Conventions

There are obviously several different methods of reading in an object module, linking it, and finally outputting the load module. The medium used could be memory only, input from and output to cassette tapes, input from and output to floppy disk, input from tape and output to disk, etc. Included in this section on interfacing are sample IO routines for tape to tape and disk to disk systems. This section assumes that the loader has itself been prepared for your system using the "bootstrap" version of the loader as described in Appendix C, Notes from a User: Implementation of Link68.

Looking at the listings of the IO tape and disk routines given in Appendices J and K, notice the various entry points (such as TABLES, OUTB, WREOF, etc.) declared at the beginning. (These same symbols are declared as External in the main loader program.) These are the names of the IO routines which the user must supply for his (her) own system. Note that some of the disk routines are supplied by the author's ICOM Floppy Disk Operating System (FDOS), while for the tape version all of the routines had to be written from scratch. Again, this may or may not be similar to the user's situation depending on the user's system configuration and software. The routines supplied in the cassette tape example could serve as a basis for any routines needed by the user.

Finally, the user should be aware that the actual length of this linking loader and all additional tables and routines as given throughout this book assume the use of the cassette tape IO routines given in Appendix J. This means that if the user supplies his (her) own routines, the lengths and capacities described elsewhere in this book may be affected.

## Tape Driver Routines

The following routines are part of a sample tape driver package. They handle the IO functions for a dual cassette tape system.

### T1INZ

This routine is used to initialize and start cassette tape for an input operation.

Called By: RDBUF
Calls: TDELY

### T1GET

This routine is used to read a character from the input tape (Tape 1). It checks for read errors and returns the error code in register B. If register B contains a 00 there were no errors.

Called By: RDBUF
Calls: none

### T1ISTP

This routine is used to stop Tape 1 after an input operation.

Called By: RDBUF
Calls: none

### T2OTZ

This routine is used to initialize and start Tape 2 for an output operation.

Called By: WRITBF
Calls: TDELY

### T2OUT

This routine is used to output a character to Tape 2. The character to be written is in register A.

Called By: WRITBF, T2OSTP
Calls: none

### T2OSTP

This routine is used to stop Tape 2 after a write operation.

Called By: WRITBF
Calls: T2OUT

### RDBUF (Tape)

This routine reads in blocks of source code from the input tape (Tape 1). It places the block of source code in INBUF. On return from this routine, the Index register points to the first location in the input buffer (INBUF)

Called By: GETB
Calls: INEEE, PDATA1, T1GET, T1INZ, T1ISTP

### WRITBF (Tape)

This routine writes out blocks of object code to Tape 2 from the output buffer. The variable OTPTR contains the

address of the last byte to be written out when the routine is called, and contains the address of the first byte in the output buffer when the routine returns.

Called By: OUTB, WREOF
Calls: T2OTZ, T2OSTP, T2OUT

## Disk Driver Routines

The disk driver routines are all in the bootstrap erasable read only memory included in the ICOM Floppy Disk Operating System (FDOS):

RIX – Read a byte from the disk, placing it in the A register.
WRT – Write a byte to the disk from the A register. The carry flag is set if End of File (EOF).
UPDATE – Close an output file.
FDOS – Load the Floppy Disk Operating System and pass control to it.

## Execution of the Linking Loader

These instructions are written assuming two different ways to load and execute the Linking Loader, depending on whether the object code for the Loader itself and the object code of the target program are on cassette tape or diskette. The main difference is the necessity of the ICOM Floppy Disk Operating System (FDOS) for the diskette. The procedures would be similar for any tape or disk system other than the two mentioned.

### Cassette Tape Files

To load the Linking Loader (LINK68) from the cassette tape is easily accomplished when the object code for it is stored in absolute Motorola MIKBUG object code format on the typical system. Using the MIKBUG "L" function loads the Linking Loader from tape. The MIKBUG "M" function sets the entry point of LINK68 (3000 hexadecimal) into memory locations A048 and A049 (hexadecimal). Note that using the "M" function merely sets up a jump address for the start of the Loader. If MIKBUG is not being used as a monitor, this may be accomplished in other ways.

LINK68 executes as a one pass linking loader, reading the input target program object code from the cassette once, and optionally placing the linked load module onto a second cassette tape. The object code tape would go in the first cassette recorder, the load module tape in the second tape machine. Ready the cassette tape in tape drive 1 for a read operation.

After this setup, using the MIKBUG "G" function begins execution of the Linking Loader, which starts by requesting:

ENTER BASE ADDRESS:

Enter the 4 hexadecimal digit base address at which the load module is to begin. Next:

ENTER UPPER MEMORY LIMIT:

which is a request for the last memory location in the user's system that is available for loading. Finally:

ENTER START OF COMMON:

where the beginning address of the memory being used for the Common area is entered. LINK68 then reads, loads, and links the input file.

When an End of File (EOF) condition is encountered on the input tape, the Linking Loader types:

EOF:REPOSITION TAPE AND TYPE CR OR TYPE A 'D' IF DONE

If there are more modules to be loaded and linked, place the appropriate cassette tape into the input drive and set the controls for a read operation. Then type a carriage return (CR). The loader then loads the new file. When there are no more files to be loaded, type a "D".

When LINK68 completes loading and linking, it types the load limits of the new load module:

LOAD LIMITS:
"XXXX YYYY"

where "XXXX" is the first memory location of the load module and the "YYYY" is the last memory location of the load module. LINK68 then lists the Load Map (see figure 1) consisting of all the Entry symbols and their associated loaded addresses.

The Linking Loader's final message is to enquire:

SAVE LOADED FILE? 'Y' OR 'N'

If the load module is to be saved, place a tape into tape drive 2 and set the controls for a write operation. Then type "Y". LINK68 saves the load module in the standard Motorola MIKBUG load format and returns control to the system monitor. If "N" is typed in response to the prompt, the load module is *not* saved, and control returns directly to the system monitor.

If there has been any tape read errors in the above process, the Linking Loader messages:

TAPE ERROR

and stops the input tape. The user should reposition the tape to the beginning of the block that produced the error and type a carriage return (CR). LINK68 will then attempt to reread the tape.

Note that the loaded program can now be executed by using the Motorola MIKBUG function "M" to place the module's entry address into memory locations A048 and A049 (hexadecimal) and then using the "G" function to begin execution.

### Diskette Files

The Linking Loader is located on diskette under the name "LINK" and is loaded and executed using the ICOM Floppy Disk Operating System (FDOS) command "RUNGO". But before the Linking Loader can be executed, the input object modules should be merged into one file by using the "MERGE" command.

Example:
MERGE,%TEST,PROG1,PROG2,PROG3

This merges the 3 programs PROG1, PROG2, and PROG3 into one file named "%TEST." %TEST would then be the input to the Linking Loader.

Example:

RUNGO,LINK,%TEST,TEST

This loads LINK, opens %TEST for input, opens TEST for output, and executes LINK.

The Linking Loader begins execution by requesting the base address of the load module:

ENTER BASE ADDRESS:

to which the user enters the 4 hexadecimal digit base address at which the load module is to begin. Next:

ENTER UPPER MEMORY LIMIT:

which is a request for the last memory location in the user's system that is available for loading. And finally:

ENTER START OF COMMON:

where the beginning address of the memory being used for the Common area is entered. LINK68 then reads, loads, and links the input file.

When the Linking Loader completes loading and linking, it types the load limits of the new load module:

LOAD LIMITS:
"XXXX YYYY"

where "XXXX" is the first memory location of the load module and "YYYY" is the last location of the load module. LINK68 then lists the Load Map (see figure 1) consisting of all the Entry symbols and their associated loaded addresses.

The Linking Loader's final message is to enquire:

SAVE LOADED FILE? 'Y' OR 'N'

If the load module is to be saved, type a "Y" and LINK68 saves the file under the name given to the output file when LINK68 was executed. Control then returns to the disk operating system. If "N" is typed in response to the prompt, the load module is *not* saved and control returns directly to the disk operating system.

Note that the loaded program can now be executed by using the Motorola MIKBUG function "M" to place the module's entry address into memory locations A048 and A049 (hexadecimal), and then using the "G" function to begin execution.

## Input Relocatable File Formats

The relocatable input file contains all of the information needed by the Linking Loader to process and load files. There are six different types of information present in the input file:

1. Object code ---------- HHHHHHHHHH.....H
2. Relocatable address fields -- HHHHR
3. Common address fields ---- HHHHM
4. Name fields ---------- HHHHPSSSSSSHHHHRN
   a b c
5. Entry fields ---------- SSSSSSHHHHRN
   d e
6. External fields ---------- SSSSSSX
   f

where "SSSSSS" is a 6 character Symbol

H is a hexadecimal character representing half a byte of object code
R is the relocation indicator
M is the Common indicator
P is the Program indicator
N is the Entry indicator
X is the External indicator

and

a is the Common length
b is the Program name
c is the Entry value
d is the Entry symbol
e is the Entry value
f is the External symbol.

## RELOCATABLE INPUT TAPE FORMAT

The Linking Loader input file is object code prepared by the Relocatable Macro Assembler and when recorded on audio cassette tape it is arranged in blocks. The maximum length (n+2, where n is the length of the object code) is set by the output buffer in the Macro Assembler. It is normally set to 512 bytes. The format is:

| | |
|---|---|
| Bytes 1 to n | Relocatable object code and information for the Linking Loader |
| Byte n+1 | End of Transmission Block (ETB) (17 hexadecimal) |
| Byte n+2 | Checksum character byte; it is the one's compliment of the summation of bytes 1 to n. |

The last block on the tape is followed by an End of File (EOF) block; it contains only one byte, an EOF character (04 hexadecimal).

# APPENDICES

# Appendix A: Error Messages

Apart from the tape error messages already discussed, the Linking Loader provides the following error messages:

SYMBOL TABLE OVERFLOW –
The Linking Loader's Symbol Table has been filled up. Reduce the number of External/Entry symbols or add additional space to the Linking Loader's Symbol Table.

****** MEMORY OVERRUN ****** –
There was not enough memory available to load the entire program.

REDEFINED
UNRESOLVED –
For each entry listed in the Load Map, one or both of these error messages may be printed if the symbol is not resolved or is defined in more than one module.

# Appendix B: Capacities

| | |
|---|---|
| Linking Loader (total) | 2 K |
| Linking Loader (actual code) | 1.5 K |
| Cassette Tape IO Routines | .5 K |
| Symbol Table (SYMTAB) | 75 entries, 9 bytes per entry |

# APPENDIX C

## Notes from a User: Implementation of Link68

*by Walter Banks, University of Waterloo*

Implementation of Link68 is accomplished by a bootstrap procedure which ultimately results in a linker specifically tailored to a unique system. This is accomplished with the use of two absolute modules presented in Appendices D and F.

In normal use RA6800ML generates relocatable object modules which are linked together by LINK68 to form a load module of absolute code. The linker itself is generated as a relocatable load module requiring linking with input and output drivers to form a usable load module. This has been overcome with the use of two absolute load modules found in Appendices D and F. The LINKER load module contains a copy of the linker, linked to location $0100 without any external references satisfied. The overlay modules contain external reference code for use with a standard MIKBUG-based system. This overlay is designed to facilitate easy initial implementation of LINK68 and serve as a template for user developed software.

The linker calls external routines through the use of a jump table which starts at location $0106. Subroutine calls within the linker go through the jump table to the overlayed routines and control is returned to the linker with an RTS instruction.

The IO structure of LINK68 assumes four separate data paths. INCH and OUTCH are input and output byte routines to the user console device. INB and OUTB are communication paths from the linker to mass storage devices such as disk, tape, or paper tape. They are used to load the relocatable modules for linking and output absolute code modules.

The jump table calls GETB which is a subroutine used to get data from a relocatable object input stream. The overlay prompts users to load new tapes when end-of-tape is sensed.

The calls to MONTOR and UPDATE are used to return control to the user supervisor program. UPDATE expects the user routine to close all open files. MONTOR is a direct entry to the user supervisor.

INITIO calls a routine which initializes IO devices and drivers. It is not needed in the simple overlay; however, room is left for a subroutine jump to a new program.

WROEOF writes an end-of-file ($04) to the output data stream.

An exception to the use of the jump table is the reference to TABLES. TABLES is used as a pointer to a data area of memory and is used only as a pointer. It must be noted that the first two locations in memory pointed to by TABLES must contain the address of TABLES+2.

Users can load a simple version of the linker by loading the absolute code module found in bar code form in Appendix E. The overlay package may be loaded on top of the linker and the combined code can be dumped to a convenient mass storage device such as a floppy disk or cassette tape. Future modifications can be made in two ways. First, the overlay package can be tailored to the unique requirements of a particular system. The absolute code may be dumped generating a new load module. Second, the whole package of linker and overlay can be linked from object files and a new load module generated.

# APPENDIX D

## LINK68 Assembly Language Object Code in Absolute Hexadecimal Format

The listing below gives the absolute object code for the linking loader LINK68 in hexadecimal format. This listing can be used to manually load the program or to verify entry of the program via the PAPER-BYTE™ bar code representation in Appendix E. Note that each line in this listing does not correspond directly to the variable length records of the bar codes, but uses a fixed length of 16 data bytes per line. The data is preceded by a 2 byte address field. Note that this program begins at hexadecimal 0100. Information on how to use this version of the linking loader to bootstrap LINK68 for the first time is given in Appendix C, with Appendix F giving details of IO routines appropriate for the bootstrap process.

```
0100 8E A0 42 7E 01 47 7E FF FF 7E FF FF 7E FF FF 7E
0110 FF FF 7E FF FF 7E FF FF 7E FF FF 7E E1 AC 7E E1
0120 D1 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0130 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0140 00 00 00 00 00 00 00 BD 01 15 FE 01 07 86 A3 C6
0150 02 AB 01 E9 00 B7 01 2F F7 01 2E EE 00 FF 01 2C
0160 FF 01 2A 7F 01 23 BD 05 5E CE 05 7F BD 05 34 BD
0170 04 FA BD 05 5E FF 01 21 FF 01 41 CE 05 94 BD 05
0180 34 BD 04 FA FF 01 45 BD 05 5E CE 05 AF BD 05 34
0190 BD 04 FA BD 05 5E FF 01 3B FF 01 3D FF 01 3F FE
01A0 01 21 BD 01 0C 25 52 81 00 27 F7 81 50 26 03 7E
01B0 02 7B 81 4E 26 03 7E 02 C5 81 58 26 03 7E 03 32
01C0 81 4D 26 03 7E 02 AA 81 52 26 03 7E 02 60 80 30
01D0 81 09 2F 02 80 07 7D 01 23 26 0C 48 48 48 48 B7
01E0 01 24 73 01 23 20 BB F6 01 24 1B A7 00 FF 01 43
01F0 08 BD 04 E1 73 01 23 20 A9 BD 05 5E BD 05 5E CE
0200 05 EC BD 05 34 BD 05 5E CE 01 41 BD 05 44 CE 01
0210 43 BD 05 44 BD 05 5E BD 05 5E BD 04 42 BD 05 5E
0220 CE 05 C7 BD 05 34 BD 05 5E CE 01 3B BD 05 44 FE
0230 01 3F BC 01 3B 27 04 09 FF 01 3F CE 01 3F BD 05
0240 44 BD 05 5E BD 05 5E CE 06 18 BD 05 34 BD 01 1B
0250 81 59 26 06 BD 05 5E 7E 06 38 BD 05 5E 7E 01 09
0260 09 09 A6 01 E6 00 BB 01 22 F9 01 21 A7 01 E7 00
0270 08 BD 04 E1 08 BD 04 E1 7E 01 A2 09 09 FF 01 21
0280 A6 01 E6 00 BB 01 3C F9 01 3B B7 01 3E F7 01 3D
0290 B6 01 40 F6 01 3F B0 01 3E F2 01 3D 24 06 FE 01
02A0 3D FF 01 3F FE 01 21 7E 01 A2 09 09 A6 01 E6 00
02B0 BB 01 3C F9 01 3B A7 01 E7 00 08 BD 04 E1 08 BD
02C0 04 E1 7E 01 A2 C6 06 F7 01 29 09 09 A6 00 B7 01
02D0 25 A6 01 B7 01 26 09 09 09 09 09 09 FF 01 43 FF
02E0 01 27 BD 03 D5 C1 FF 26 0E BD 03 95 FE 01 30 6C
02F0 08 FE 01 43 7E 01 A2 C5 01 27 09 CA 80 FE 01 30
0300 E7 08 20 ED FF 01 27 B6 01 25 F6 01 26 FE 01 30
0310 6C 08 A7 06 E7 07 FE 01 27 EE 00 FF 01 30 FE 01
0320 27 A7 00 E7 01 FE 01 30 8C FF FF 27 C4 FF 01 27
0330 20 E7 C6 06 F7 01 29 09 09 09 09 09 09 FF 01 27
0340 FF 01 25 08 08 FF 01 43 BD 03 D5 C1 FF 26 12 BD
0350 03 95 FE 01 25 86 FF A7 00 A7 01 FE 01 43 7E 01
0360 A2 FE 01 30 E6 08 C5 01 27 0D A6 06 E6 07 FE 01
0370 25 A7 00 E7 01 20 E4 EE 06 FF 01 27 EE 00 8C FF
0380 FF 26 F6 B6 01 25 F6 01 26 FE 01 27 A7 00 E7 01
0390 FE 01 25 20 C0 FE 01 2A FF 01 30 BC 01 2E 26 09
03A0 CE 05 69 BD 05 34 7E 01 09 FE 01 27 A6 00 08 FF
03B0 01 27 FE 01 2A A7 00 08 FF 01 2A 7A 01 29 26 E9
03C0 B6 01 25 A7 00 B6 01 26 A7 01 86 00 A7 02 08 08
03D0 08 FF 01 2A 39 FE 01 2C BC 01 2A 27 2B FF 01 30
03E0 FF 01 32 FE 01 27 FF 01 34 B6 01 29 B7 01 36 CE
03F0 01 32 BD 04 13 27 14 FE 01 30 08 08 08 08 08 08
0400 08 08 08 BC 01 2A 26 D5 C6 FF 39 FE 01 30 E6 08
0410 EE 06 39 36 37 E6 04 FF 01 37 FE 01 37 EE 00 A6
0420 00 FE 01 37 6C 01 26 02 6C 00 FE 01 37 EE 02 A1
```

```
0430 00 26 0C FE 01 37 6C 03 26 02 6C 02 5A 26 DB 33
0440 32 39 CE 05 F9 BD 05 34 BD 05 5E FE 01 2C BC 01
0450 2A 27 56 CE FF FF FF 01 39 FE 01 2C FF 01 30 86
0460 FF A1 08 27 1C 08 08 08 08 08 08 FF 01 32 CE 01
0470 39 FF 01 34 86 02 B7 01 36 CE 01 32 BD 04 13 25
0480 13 FE 01 30 08 08 08 08 08 08 08 08 08 BC 01 2A
0490 27 0F 20 C8 FE 01 30 FF 01 27 EE 06 FF 01 39 20
04A0 E0 CE FF FF BC 01 39 26 01 39 FE 01 27 C6 06 A6
04B0 00 08 BD 01 1E 5A 26 F7 BD 05 48 BD 05 44 E6 00
04C0 C5 01 26 06 CE 06 02 BD 05 34 C5 80 27 06 CE 06
04D0 0E BD 05 34 BD 05 5E FE 01 27 86 FF A7 08 7E 04
04E0 53 BC 01 45 27 01 39 BD 05 5E CE 05 CF BD 05 34
04F0 BD 05 5E 31 31 7E 01 F9 00 00 8D 0C B7 04 F8 8D
0500 07 B7 04 F9 FE 04 F8 39 8D 09 48 48 48 48 16 8D
0510 02 1B 39 BD 01 1B 80 30 2B 0F 81 09 2F 0A 81 11
0520 2B 07 81 16 2E 03 80 07 39 86 3F BD 01 1E 20 E3
0530 BD 01 1E 08 A6 00 81 04 26 F6 39 A6 00 8D 0E A6
0540 00 08 20 0D 8D F5 8D F3 86 20 7E 01 1E 44 44 44
0550 44 84 0F 8B 30 81 39 23 02 8B 07 7E 01 1E 86 0D
0560 BD 01 1E 86 0A BD 01 1E 39 53 59 4D 42 4F 4C 20
0570 54 41 42 4C 45 20 4F 56 45 52 46 4C 4F 57 04 45
0580 4E 54 45 52 20 42 41 53 45 20 41 44 44 52 45 53
0590 53 3A 20 04 45 4E 54 45 52 20 55 50 50 45 52 20
05A0 4D 45 4D 4F 52 59 20 4C 49 4D 49 54 3A 20 04 45
05B0 4E 54 45 52 20 53 54 41 52 54 20 4F 46 20 43 4F
05C0 4D 4D 4F 4E 3A 20 04 43 4F 4D 4D 4F 4E 3A 04 2A
05D0 2A 2A 2A 2A 20 4D 45 4D 4F 52 59 20 4F 56 45
05E0 52 52 55 4E 20 2A 2A 2A 2A 2A 2A 04 4C 4F 41 44
05F0 20 4C 49 4D 49 54 53 3A 04 4C 4F 41 44 20 4D 41
0600 50 04 55 4E 52 45 53 4F 4C 56 45 44 20 04 52 45
0610 44 45 46 49 4E 45 44 04 53 41 56 45 20 4C 4F 41
0620 44 45 44 20 46 49 4C 45 3F 20 22 59 22 20 4F 52
0630 20 22 4E 22 20 04 00 00 B6 01 44 B0 01 42 F6 01
0640 43 F2 01 41 26 04 81 10 25 02 86 0F 8B 04 B7 06
0650 36 80 03 B7 06 37 CE 06 C4 BD 06 B6 5F CE 06 36
0660 8D 33 CE 01 41 BD 06 95 BD 06 95 FE 01 41 BD 06
0670 95 7A 06 37 26 F8 FF 01 41 53 37 30 BD 06 95 33
0680 FE 01 41 09 BC 01 43 26 AF CE 06 BD BD 06 B6 BD
0690 01 12 7E 01 18 EB 00 A6 00 8D 05 A6 00 08 20 04
06A0 44 44 44 44 84 0F 8B 30 81 39 23 02 8B 07 BD 01
06B0 0F 39 BD 01 0F 08 A6 00 81 04 26 F6 39 0D 0A 53
06C0 39 0D 0A 04 0D 0A 53 31 04
```

# APPENDIX E

## PAPERBYTE™ Bar Code Representation of Link68 in Absolute Format

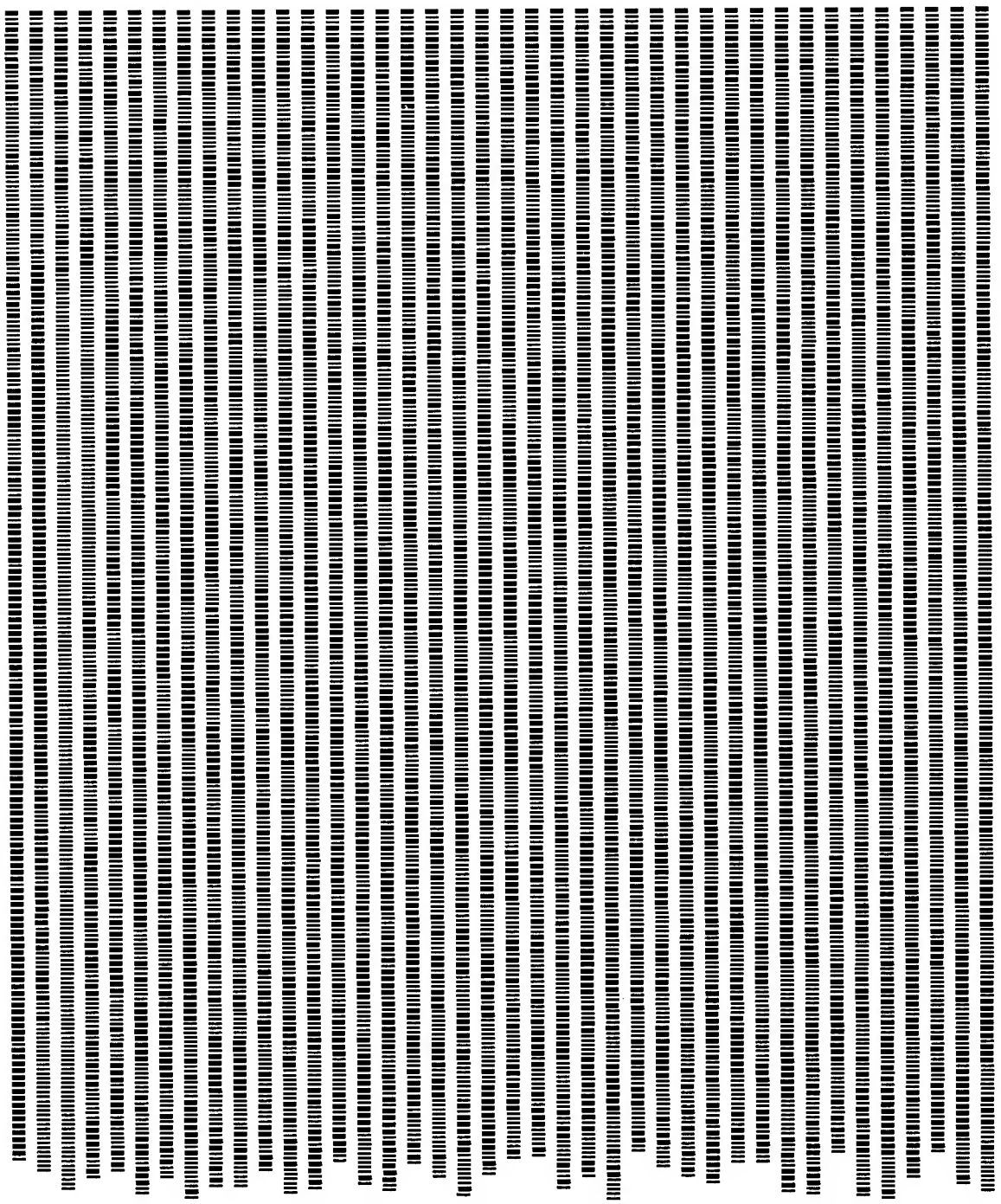
Beginning on the following page is a complete machine readable representation (PAPERBYTE™ bar codes) of the object code for Grappel and Hemenway's linking loader LINK68. The object code was created by assembling LINK68 using the relocatable macro assembler, available as the PAPERBYTE™ book RA6800ML: An M6800 Relocatable Macro Assembler (ISBN 0-931718-10-4). See Appendix G for a listing of the 6800 assembly language source code of the linking loader.

This representation uses the absolute loader format, in which each bar code frame (one line of bar codes running from top to bottom of the page) contains a 2 byte address followed by data which is loaded in ascending order starting at that address. A hexadecimal listing that can be used to verify the input from bar codes is given in Appendix D. For details on the frame format and absolute loader format used in this and other PAPERBYTE™ books, see PAPERBYTE publication *Bar Code Loader* by Ken Budnick. The book contains a brief history on bar codes, a general bar code loader algorithm with flowcharts, and complete program listings for 6800, 6502, and 8080 or Z-80 based systems.

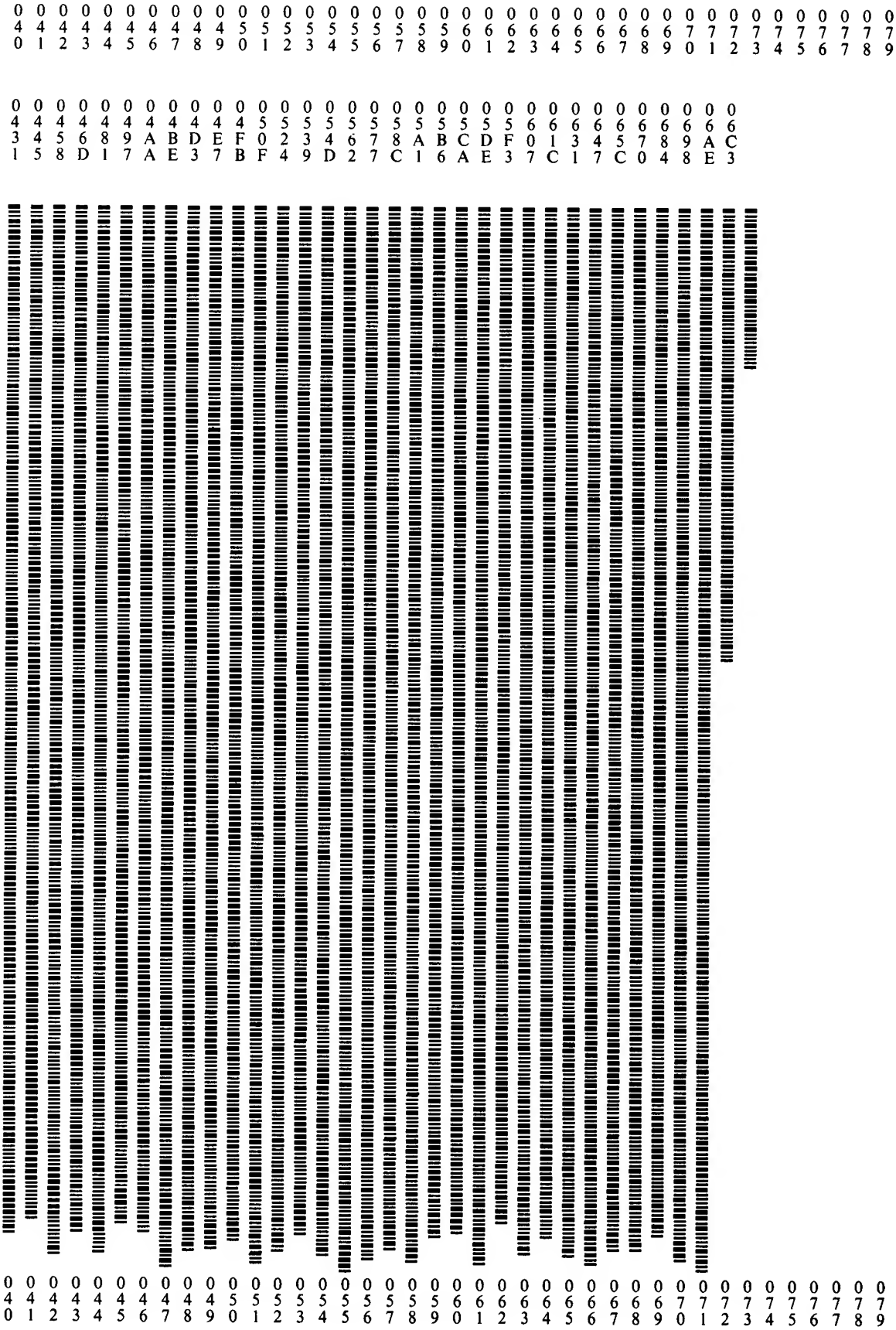
Information on how to use this version of the linking loader to bootstrap LINK68 for the first time is given in Appendix C, with Appendix F giving details of IO routines appropriate for the bootstrap process.

0000 0001 0002 0003 0004 0005 0006 0007 0008 0009 0010 0011 0012 0013 0014 0015 0016 0017 0018 0019 0020 0021 0022 0023 0024 0025 0026 0027 0028 0029 0030 0031 0032 0033 0034 0035 0036 0037 0038 0039

0100 0112 0125 013E 0154 0168 017C 0190 01A4 01B9 01CE 01E3 01F8 020C 0220 0234 0248 025C 0271 0286 029A 02AE 02C2 02D7 02EB 0300 0314 0328 033C 0350 0364 0378 038C 03A1 03B5 03CA 03DF 03F3 0408 041C



0000 0001 0002 0003 0004 0005 0006 0007 0008 0009 0010 0011 0012 0013 0014 0015 0016 0017 0018 0019 0020 0021 0022 0023 0024 0025 0026 0027 0028 0029 0030 0031 0032 0033 0034 0035 0036 0037 0038 0039


040 041 042 043 044 045 046 047 048 049 050 051 052 053 054 055 056 057 058 059 060 061 062 063 064 065 066 067 068 069 070 071 072 073 074 075 076 077 078 079
043 045 046 048 04A 04B 04D 04E 04F 050 052 053 054 055 057 058 05A 05B 05C 05D 05F 060 061 063 064 065 067 068 069 06A 06C

# APPENDIX F

## Input and Output Routines for LINK68 in Absolute Format with PAPERBYTE™ Bar Code Representation

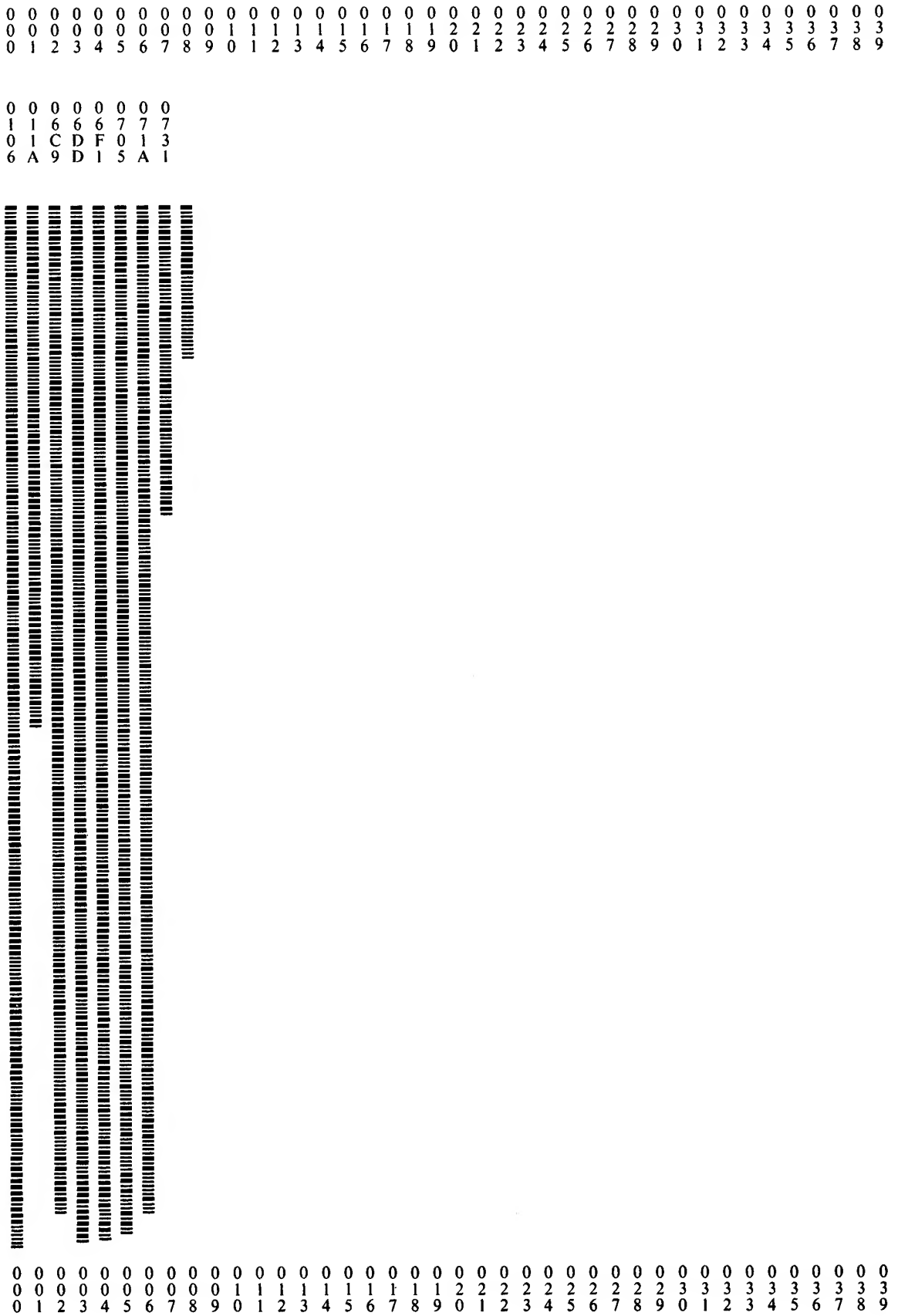
These overlay modules contain external reference code to the linking loader LINK68 for use with a standard MIKBUG-based system. This overlay is designed to facilitate easy initial implementation of LINK68 and serve as a template for user developed software. These routines can be used in conjunction with the version of LINK68 given in Appendices D and E to bootstrap LINK68 for the first time. Details of the bootstrap process are given in Appendix C.

On page 29 is the machine readable representation (PAPERBYTE™ bar codes) of the object code of the IO routines listed below. The representation uses the absolute loader format, in which each bar code frame (one line of bars running from top to bottom of the page) contains a 2 byte address followed by data which is loaded in ascending order starting at that address.

For details on the frame and absolute loader format used in this and all PAPERBYTE™ books, see the PAPERBYTE publication *Bar Code Loader* by Ken Budnick. This book contains a brief history on bar codes, a general bar code loader algorithm with flowcharts, and complete program listings for 6800, 6502, and 8080 or Z-80 based systems.

```
00001                          NAM    LINKIO

00003        0100    START     EQU    $0100     START OF THE LINKER

00005        E1AC    INCH      EQU    $E1AC     INPUT CHAR (MIKBUG)
00006        E1D1    OUTCH     EQU    $E1D1     OUTPUT CHAR (MIKBUG)
00007        E1AC    INB       EQU    $E1AC     INPUT DATA CHAR TO LINKER
00008        E1D1    OUTB      EQU    $E1D1     OUT DATA CHAR FROM LINKER
00009        E0E3    MONTOR    EQU    $E0E3     EXIT BACK TO MONITOR (MIKBUG)
00010        E0E3    UPDATE    EQU    $E0E3     CLOSE OUTPUT FILES ,EXIT

00012 0106                     ORG    START+6

00014 0106 7E 0731             JMP    TABLES    START OF SYMBOL TABLE
00015 0109 7E E0E3             JMP    MONTOR    MONITOR START ADDRESS
00016 010C 7E 06CD             JMP    GETB      READ A BYTE FROM RELOCATION
00017                  *                          INPUT STRING
00018 010F 7E E1D1             JMP    OUTB      WRITE A BYTE
00019 0112 7E 06FA             JMP    WREOF     WRITE EOF ON SAVE FILE
00020 0115 7E 06C9             JMP    INITIO    INIT IO DEVICES
00021 0118 7E E0E3             JMP    UPDATE    CLOSE AN OUTPUT FILE
00022 011B 7E E1AC INEEE       JMP    INCH      INPUT CHAR TO ACC A FROM
00023                  *                          CONSOLE
00024 011E 7E E1D1 OUTEEE      JMP    OUTCH     OUTPUT BYTE IN ACC A

00026        0534    PDAT1     EQU    START+$0434    PRINT CHAR STRING
00027        055E    CRLF      EQU    START+$045E    PRINT <CR> <LF>

00029 06C9                     ORG    START+$05C9    START AT THE END OF
00030                  *                                THE LINKER

00032 06C9 01          INITIO  NOP           INITIALIZE      I/O DRIVERS
00033 06CA 01                  NOP
00034 06CB 01                  NOP
00035 06CC 39                  RTS

00037 06CD FF 072F GETB        STX    DXSV      SAVE INDEX REGISTER
00038 06D0 BD E1AC GET1        JSR    INB       INPUT A DATA CHARACTER
00039 06D3 81 04               CMP A  #$04      IS IT END OF FILE
00040 06D5 26 16               BNE    XIT       NO EXIT
00041 06D7 CE 0700             LDX    #EOF      YES PRINT EOF MESSAGE ON
00042                  *                          CONSOLE
00043 06DA BD 0534             JSR    PDAT1
00044 06DD BD 011B RD6         JSR    INEEE     FOR CONSLE RESOPONSE
00045 06E0 81 0D               CMP A  #$0D      <CR> START READING NEXT TAPE
```

```
00046 06E2 27 EC          BEQ    GET1
00047 06E4 81 44          CMP A  #'D       D THIS WAS THE LAST TAPE
00048 06E6 26 F5          BNE    RD6
00049 06E8 FE 072F        LDX    DXSV      RESTORE INDEX
00050 06EB 0D             SEC              SET     CARRY END OF FILE
00051 06EC 39             RTS
00052 06ED 81 0D   XIT     CMP A  #$0D
00053 06EF 27 DF          BEQ    GET1
00054 06F1 81 2F          CMP A  #'/       STRIP OFF CONTROL CHARACTERS
00055 06F3 2D DB          BLT    GET1
00056 06F5 FE 072F        LDX    DXSV      RESTORE INDEX REGISTER
00057 06F8 0C             CLC              CLEAR   CARRY NOT EOF
00058 06F9 39             RTS

00060 06FA 96 04   WREOF   LDA A  4         LOAD ASCII EOF
00061 06FC BD E1D1        JSR    OUTB      OUTPUT IT TO DATA STREAM
00062 06FF 39             RTS

00064 0700 0D0A    EOF     FDB    $0D0A     <CR><LF>
00065 0702 454F46         FCC    /EOF: NEXT TAPE,TYPE CR/
      0705 3A204E
      0708 455854
      070B 205441
      070E 50452C
      0711 545950
      0714 452043
      0717 52
00066 0718 0D0A           FDB    $0D0A     <CR><LF>
00067 071A 545950         FCC    /TYPE   "D" IF DONE/
      071D 452020
      0720 202244
      0723 222049
      0726 462044
00068 072C 0D0A           FDB    $0D0A     <CR><LF>
00069 072E 04             FCB    4

00071 072F 0002    DXSV    RMB    2         SAVE SPACE FOR TEMP STORAGE OF
00072              *                         THE INDEX REGISTER

00074 0731 0733    TABLES  FDB    *+2       START OF SYMBOL TABLE

00076                      END
```


000 001 002 003 004 005 006 007 008 009 010 011 012 013 014 015 016 017 018 019 020 021 022 023 024 025 026 027 028 029 030 031 032 033 034 035 036 037 038 039
0106 011A 06C9 06DD 06F1 0705 071A 0731

# APPENDIX G

## Assembly Language Source Listing of LINK68

This assembly was executed using the relocatable macro assembler RA6800ML available in the PAPER-BYTE™ book *RA6800ML: An M6800 Relocatable Macro Assembler* by Jack Hemenway (ISBN 0-931718-10-4). The object code in the assembly listing can be used without relocation if the program is loaded at location zero (hexadecimal) in memory. When creating a final object module for the loader, hand entered overlays for the Motorola MIKBUG monitor or the ICOM Floppy Disk Operating System IO routines will be necessary. The routines given in Appendices J and K can be used directly with their respective operating system, or as guidelines for coding patches to interface the above systems or other monitor programs.

```
0000 0000    N          NAM LINK    LINKING LOADER
                        *
                        *       C COPYRIGHT 1977 BY
                        *       ROBERT D. GRAPPEL LEXINGTON MASS.
                        *       AND JACK E. HEMENWAY BOSTON MASS.
                        *       ALL RIGHTS RESERVED
                        *
                        *
0000 8E A042            LDS #$A042
0003 7E 0047 R          JMP LOAD
                        *
0006 7E 0000 X          EXT TABLES      START OF SYMTAB
0009 7E 0000 X          EXT MONTOR      MONITOR
000C 7E 0000 X          EXT GETB        READ A BYTE
000F 7E 0000 X          EXT OUTB        WRITE A BYTE
0012 7E 0000 X          EXT WREOF       WRITE EOF
0015 7E 0000 X          EXT INITIO      INIT FOR I/O
0018 7E 0000 X          EXT UPDATE      CLOSE AN OUTPUT FILE
                        *
                        *
001B 001B    N          ENT INEEE
001B 0434    N          ENT PDATA1
001B 045E    N          ENT CRLF
                        *
                        *
001B 7E E1AC  INEEE     JMP $E1AC INPUT A CHAR
001E 7E E1D1  OUTEEE    JMP $E1D1 OUTPUT A CHAR TO TTY
0021 0002     BASE      RMB 2     BASE ADDRESS
0023 0001     NFLAG     RMB 1     NIBBLE FLAG 00=LEFT, FF=RIGHT
0024 0001     BYTE      RMB 1     TEMPORARY LOCATION
0025 0002     LC        RMB 2     LOCATION COUNTER
0027 0002     DESCRA    RMB 2     DESCRIPTOR ADDRESS
0029 0001     DESCRC    RMB 1     DESCRIPTOR COUNT
002A 0002     NXTSYM    RMB 2     NEXT ENTRY IN SYMTAB
002C 0002     SYMTAB    RMB 2     SYMBOL TABLE
002E 0002     SYMEND    RMB 2     END OF TABLE
0030 0002     SYMPTR    RMB 2     SYMTAB POINTER
0032 0002     STRNG1    RMB 2     PARM LIST
0034 0002     STRNG2    RMB 2     FOR
0036 0001     COUNT     RMB 1     COMPAR
0037 0002     XSAV      RMB 2     TEMP
0039 0002     HIVAL     RMB 2     HIGHEST COMMON COUNT
003B 0002     CBAS      RMB 2     START OF COMMON
003D 0002     CBASSV    RMB 2     CBAS SAVE TEMP
003F 0002     HICBAS    RMB 2     END OF COMMON
0041 0002     BASESV    RMB 2     FIRST LOCATION
0043 0002     LAST      RMB 2     LAST LOCATION
0045 0002     UPLIM     RMB 2     UPPER MEMORY LIMIT
                        *
                        *
                        * LOAD IS THE ENTRY POINT TO THE LOADER
                        *
0047 BD 0015 R LOAD     JSR INITIO      INIT THE I/O
004A FE 0007 R          LDX TABLES+1    POINT TO START OF SYMTAB
004D 86 A3              LDA A #$A3      ADD SPACE FOR 75 SYMBOLS
004F C6 02              LDA B #$02
0051 AB 01              ADD A 1,X       ADD TO START OF SYMTAB
0053 E9 00              ADC B 0,X
0055 B7 002F R          STA A SYMEND+1      INIT
0058 F7 002E R          STA B SYMEND
005B EE 00              LDX 0,X         GET START OF SYMTAB
005D FF 002C R          STX SYMTAB      INIT
0060 FF 002A R          STX NXTSYM      INIT
                        *
0063 7F 0023 R          CLR NFLAG       NFLAG:=LEFT
0066 BD 045E R          JSR CRLF
                        *
0069 CE 047F R          LDX #MSGL       GET BASE ADDRESS
006C BD 0434 R          JSR PDATA1
006F BD 03FA R          JSR BADDR       GET VALUE IN HEX
0072 BD 045E R          JSR CRLF
0075 FF 0021 R          STX BASE        INIT
0078 FF 0041 R          STX BASESV      INIT
                        *
007B CE 0494 R          LDX #MSGA       GET UPPER MEMORY LIMIT
007E BD 0434 R          JSR PDATA1
0081 BD 03FA R          JSR BADDR       GET VALUE IN HEX
0084 FF 0045 R          STX UPLIM       INIT
0087 BD 045E R          JSR CRLF
                        *
008A CE 04AF R          LDX #MSGB       GET START OF COMMON
008D BD 0434 R          JSR PDATA1
0090 BD 03FA R          JSR BADDR       INIT
0093 BD 045E R          JSR CRLF
0096 FF 003B R          STX CBAS        INIT
0099 FF 003D R          STX CBASSV      INIT
009C FF 003F R          STX HICBAS      INIT
009F FE 0021 R          LDX BASE        GET START OF MEMORY
                        *
                        *
00A2 BD 000C R LOAD2    JSR GETB    GET A BYTE
00A5 25 52              BCS LOADE   EOF
                        *
00A7 81 00              CMP A #$00  NULL?
00A9 27 F7              BEQ LOAD2   YES
                        *
00AB 81 50              CMP A #'P   PROGRAM MODULE?
00AD 26 03              BNE *+5
                        *
00AF 7E 017B R          JMP LOADP   YES
                        *
00B2 81 4E              CMP A #'N   ENTRY ?
00B4 26 03              BNE *+5     NO
                        *
00B6 7E 01C5 R          JMP LOADN   YES
                        *
00B9 81 58              CMP A #'X   EXTERNAL ?
00BB 26 03              BNE *+5     NO
                        *
00BD 7E 0232 R          JMP LOADX   YES
                        *
00C0 81 4D              CMP A #'M   "COMMON"?
00C2 26 03              BNE *+5     NO
                        *
00C4 7E 01AA R          JMP LOADM   YES
                        *
00C7 81 52              CMP A #'R   RELOCATABLE ?
00C9 26 03              BNE *+5     NO
                        *
00CB 7E 0160 R          JMP LOADR   YES
                        *
                        * CONVERT TWO ASCII BYTES TO ONE HEX BYTE
                        *
00CE 80 30              SUB A #$30
00D0 81 09              CMP A #$09
00D2 2F 02              BLE *+4
00D4 80 07              SUB A #$07
                        *
00D6 7D 0023 R          TST NFLAG   WHICH NIBBLE ?
00D9 26 0C              BNE RNIBL   RIGHT
                        *
00DB 48                 ASL A       LEFT
00DC 48                 ASL A
00DD 48                 ASL A
00DE 48                 ASL A
00DF B7 0024 R          STA A BYTE  SAVE LEFT NIBBLE
00E2 73 0023 R          COM NFLAG   SET FOR RIGHT NIBBLE
00E5 20 BB              BRA LOAD2
                        *
00E7 F6 0024 R RNIBL    LDA B BYTE  GET LEFT NIBBLE
00EA 1B                 ABA
00EB A7 00              STA A 0,X
00ED FF 0043 R          STX LAST    LOADED ADDRESS
00F0 08                 INX
00F1 BD 03E1 R          JSR MEMCHK  CHECK MEMORY LIMIT
00F4 73 0023 R          COM NFLAG   SET FOR LEFT NIBBLE
00F7 20 A9              BRA LOAD2
                        * EOF FINISH LOAD
                        *
00F9 BD 045E R LOADE    JSR CRLF
00FC BD 045E R          JSR CRLF
```

```
00FF CE 04EC R          LDX #MSGE       "LOAD LIMITS"
0102 BD 0434 R          JSR PDATA1
0105 BD 045E R          JSR CRLF
0108 CE 0041 R          LDX #BASESV     PRINT STARTING ADDRESS
010B BD 0444 R          JSR OUT4HS
010E CE 0043 R          LDX #LAST
0111 BD 0444 R          JSR OUT4HS      PRINT END ADDRESS
0114 BD 045E R          JSR CRLF
0117 BD 045E R          JSR CRLF
                *
011A BD 0342 R          JSR PRTSYM      PRINT LOAD MAP
011D BD 045E R          JSR CRLF        "COMMON"
0120 CE 04C7 R          LDX #MSGC
0123 BD 0434 R          JSR PDATA1
0126 BD 045E R          JSR CRLF
0129 CE 003B R          LDX #CBAS       PRINT START OF COMMON
012C BD 0444 R          JSR OUT4HS
012F FE 003F R          LDX HICBAS
0132 BC 003B R          CPX CBAS        ANY COMMON?
0135 27 04              BEQ LOADE1      NO
                *
0137 09                 DEX             HICBAS:=HICBAS-1
0138 FF 003F R          STX HICBAS
                *
013B CE 003F R LOADE1   LDX #HICBAS      PRINT END OF COMMON
013E BD 0444 R          JSR OUT4HS
0141 BD 045E R          JSR CRLF
0144 BD 045E R          JSR CRLF
                *
                *
0147 CE 0518 R          LDX #SAVFIL
014A BD 0434 R          JSR PDATA1      PRINT PROMPT
014D BD 001B R          JSR INEEE
0150 81 59              CMP A #'Y       SAVE LOADED FILE?
0152 26 06              BNE *+8         NO
                *
0154 BD 045E R          JSR CRLF        YES
0157 7E 0538 R          JMP PUNCH
                *
015A BD 045E R          JSR CRLF
015D 7E 0009 R          JMP MONTOR      ALL DONE
                * RELOCATE ADDRESS
                *
0160 09         LOADR   DEX             POINT TO ADDRESS
0161 09                 DEX
0162 A6 01              LDA A 1,X       GET ADDRESS
0164 E6 00              LDA B 0,X
0166 BB 0022 R          ADD A BASE+1    ADD IN RELOCATION
0169 F9 0021 R          ADC B BASE
016C A7 01              STA A 1,X       STORE
016E E7 00              STA B 0,X
0170 08                 INX             POINT TO NEXT ADDRESS
0171 BD 03E1 R          JSR MEMCHK      CHECK MEMORY LIMIT
0174 08                 INX
0175 BD 03E1 R          JSR MEMCHK      CHECK MEMORY LIMIT
0178 7E 00A2 R          JMP LOAD2
                * PROGRAM MODULE
                *
017B 09         LOADP   DEX             BACKUP OVER COMMON LENGTH
017C 09                 DEX
017D FF 0021 R          STX BASE        SAVE AS NEW BASE
0180 A6 01              LDA A 1,X       ADD COMMON LENGTH TO CBAS
0182 E6 00              LDA B 0,X
0184 BB 003C R          ADD A CBAS+1
0187 F9 003B R          ADC B CBAS
018A B7 003E R          STA A CBASSV+1  SAVE IN CBASSV
018D F7 003D R          STA B CBASSV
                *
                * SEE IF NEW COMMON IS LONGER
                * THAN LAST COMMON
                *
0190 B6 0040 R          LDA A HICBAS+1
0193 F6 003F R          LDA B HICBAS
0196 B0 003E R          SUB A CBASSV+1
0199 F2 003D R          SBC B CBASSV
                *
019C 24 06              BCC LOADP1      NO
                *
019E FE 003D R          LDX CBASSV      YES
01A1 FF 003F R          STX HICBAS
                *
01A4 FE 0021 R LOADP1   LDX BASE        LOAD NEW BASE
01A7 7E 00A2 R          JMP LOAD2
```

```
                * COMMON RELOCATION
                *
01AA 09         LOADM   DEX             POINT TO ADDRESS
01AB 09                 DEX
01AC A6 01              LDA A 1,X       GET ADDRESS
01AE E6 00              LDA B 0,X
01B0 BB 003C R          ADD A CBAS+1    ADD IN BASE OF COMMON
01B3 F9 003B R          ADC B CBAS
01B6 A7 01              STA A 1,X       STORE
01B8 E7 00              STA B 0,X
01BA 08                 INX             POINT TO NEXT ADDRESS
01BB BD 03E1 R          JSR MEMCHK      CHECK MEMORY LIMIT
01BE 08                 INX
01BF BD 03E1 R          JSR MEMCHK      CHECK MEMORY LIMIT
                *
01C2 7E 00A2 R          JMP LOAD2
                * HANDLE ENTRY SYMBOL
                *
01C5 C6 06      LOADN   LDA B #6        6 CHARS/SYMBOL
01C7 F7 0029 R          STA B DESCRC
01CA 09                 DEX
01CB 09                 DEX
01CC A6 00              LDA A 0,X       LC:=ENTRY VALUE
01CE B7 0025 R          STA A LC
01D1 A6 01              LDA A 1,X
01D3 B7 0026 R          STA A LC+1
01D6 09                 DEX             BACK UP TO START OF SYMBOL
01D7 09                 DEX
01D8 09                 DEX
01D9 09                 DEX
01DA 09                 DEX
01DB 09                 DEX
01DC FF 0043 R          STX LAST        INIT LAST
01DF FF 0027 R          STX DESCRA      POINT TO SYMBOL
01E2 BD 02D5 R          JSR LKPSYM
01E5 C1 FF              CMP B #$FF      SYMBOL IN TABLE ?
01E7 26 0E              BNE LOADN4      YES
                *
01E9 BD 0295 R          JSR STOSYM      STORE SYMBOL
01EC FE 0030 R          LDX SYMPTR
01EF 6C 08              INC 8,X         SET DEFINED BIT
                *
01F1 FE 0043 R LOADN3   LDX LAST        RESTORE POINTER
01F4 7E 00A2 R          JMP LOAD2
                *
01F7 C5 01      LOADN4  BIT B #$01      ALREADY DEFINED ?
01F9 27 09              BEQ LOADN6      NO
                *
01FB CA 80              ORA B #$80      SET REDEFINED BIT
01FD FE 0030 R          LDX SYMPTR
0200 E7 08              STA B 8,X
0202 20 ED              BRA LOADN3
                *
0204 FF 0027 R LOADN6   STX DESCRA
0207 B6 0025 R          LDA A LC        ADDRESS OF SYMBOL
020A F6 0026 R          LDA B LC+1
020D FE 0030 R          LDX SYMPTR
0210 6C 08              INC 8,X         SET DEFINED BIT
0212 A7 06              STA A 6,X
0214 E7 07              STA B 7,X
                *
                * FOLLOW LINKAGES
                *
0216 FE 0027 R          LDX DESCRA
0219 EE 00      LOADN5  LDX 0,X
021B FF 0030 R          STX SYMPTR
021E FE 0027 R          LDX DESCRA
0221 A7 00              STA A 0,X
0223 E7 01              STA B 1,X
0225 FE 0030 R          LDX SYMPTR
0228 8C FFFF            CPX #$FFFF      AT END LINK ?
022B 27 C4              BEQ LOADN3      YES
                *
022D FF 0027 R          STX DESCRA      NO
0230 20 E7              BRA LOADN5
                * HANDLE EXTERNAL SYMBOL
                *
0232 C6 06      LOADX   LDA B #6        6 CHARS/SYMBOL
0234 F7 0029 R          STA B DESCRC
0237 09                 DEX             BACK UP TO START OF SYMBOL
0238 09                 DEX
0239 09                 DEX
023A 09                 DEX
023B 09                 DEX
023C 09                 DEX
023D FF 0027 R          STX DESCRA      POINT TO SYMBOL
0240 FF 0025 R          STX LC          SAVE ADDRESS
0243 08                 INX
0244 08                 INX
0245 FF 0043 R          STX LAST        SAVE ADDRESS
0248 BD 02D5 R          JSR LKPSYM
024B C1 FF              CMP B #$FF      IN TABLE ?
024D 26 12              BNE LOADX4      YES
                *
024F BD 0295 R          JSR STOSYM      NO,SAVE SYMBOL
```

```
0252 FE 0025 R          LDX LC
0255 86 FF      LOADX2  LDA A #$FF      SET END LINK
0257 A7 00              STA A 0,X
0259 A7 01              STA A 1,X
                *
025B FE 0043 R LOADX3   LDX LAST        LOAD NEW LAST
025E 7E 00A2 R          JMP LOAD2
                *
0261 FE 0030 R LOADX4   LDX SYMPTR      POINT TO SYMBOL ENTRY
0264 E6 08              LDA B 8,X       GET INFO BYTE
0266 C5 01              BIT B #$01      DEFINED ?
0268 27 0D              BEQ LOADX5      NO
                *
026A A6 06              LDA A 6,X       GET ENTRY ADDRESS
026C E6 07              LDA B 7,X
026E FE 0025 R          LDX LC
0271 A7 00              STA A 0,X
0273 E7 01              STA B 1,X       SAVE ADDRESS
0275 20 E4              BRA LOADX3
                *
0277 EE 06      LOADX5  LDX 6,X         GET FIRST LINK
                *
0279 FF 0027 R LOADX6   STX DESCRA      FOLLOW LINKAGE
027C EE 00              LDX 0,X
027E 8C FFFF            CPX #$FFFF      END LINK ?
0281 26 F0              BNE LOADX6      NO
                *
0283 B6 0025 R          LDA A LC
0286 F6 0026 R          LDA B LC+1
0289 FE 0027 R          LDX DESCRA
028C A7 00              STA A 0,X
028E E7 01              STA B 1,X       NEW LINK ADDRESS
0290 FE 0025 R          LDX LC
0293 20 C0              BRA LOADX2      SET NEW END LINK
                * SYMBOL TABLE ROUTINES
                *
                * STORE SYMBOL IN SYMTAB
                *
0295 FE 002A R STOSYM   LDX NXTSYM
0298 FF 0030 R          STX SYMPTR      SAVE ENTRY ADDRESS
029B BC 002E R          CPX SYMEND      FULL ?
029E 26 09              BNE STOSY1      NO
                *
02A0 CE 0469 R          LDX #SYMFUL     ERROR
02A3 BD 0434 R          JSR PDATA1
02A6 7E 0009 R          JMP MONTOR      RETURN TO EXEC
                *
                * MOVE SYMBOL TO SYMTAB
                *
02A9 FE 0027 R STOSY1   LDX DESCRA      GET ADDRESS OF SYMBOL
02AC A6 00              LDA A 0,X
02AE 08                 INX
02AF FF 0027 R          STX DESCRA
02B2 FE 002A R          LDX NXTSYM
02B5 A7 00              STA A 0,X
02B7 08                 INX
02B8 FF 002A R          STX NXTSYM
02BB 7A 0029 R          DEC DESCRC
02BE 26 E9              BNE STOSY1
                *
02C0 B6 0025 R          LDA A LC        GET LC
02C3 A7 00              STA A 0,X       STORE
02C5 B6 0026 R          LDA A LC+1
02C8 A7 01              STA A 1,X
02CA 86 00              LDA A #$00      SET INFO BIT
02CC A7 02              STA A 2,X
02CE 08                 INX
02CF 08                 INX
02D0 08                 INX
02D1 FF 002A R          STX NXTSYM
02D4 39                 RTS
                * LOOKUP SYMBOL IN SYMTAB
                *
02D5 FE 002C R LKPSYM   LDX SYMTAB
02D8 BC 002A R          CPX NXTSYM      EMPTY TABLE?
02DB 27 2B              BEQ LKPSY3      YES
                *
02DD FF 0030 R LKPSY1   STX SYMPTR
02E0 FF 0032 R          STX STRNG1
02E3 FE 0027 R          LDX DESCRA
02E6 FF 0034 R          STX STRNG2
02E9 B6 0029 R          LDA A DESCRC
02EC B7 0036 R          STA A COUNT
02EF CE 0032 R          LDX #STRNG1
02F2 BD 0313 R          JSR COMPAR
02F5 27 14              BEQ LKPSY2      MATCH
                *
02F7 FE 0030 R          LDX SYMPTR
02FA 08                 INX
02FB 08                 INX
02FC 08                 INX
02FD 08                 INX
02FE 08                 INX
02FF 08                 INX
0300 08                 INX
0301 08                 INX
0302 08                 INX
0303 BC 002A R          CPX NXTSYM      END OF ENTRYS ?
0306 26 D5              BNE LKPSY1      NO
                *
                * NOT IN SYMTAB
                *
0308 C6 FF      LKPSY3  LDA B #$FF
030A 39                 RTS
                *
                * FOUND SYMBOL
                *
030B FE 0030 R LKPSY2   LDX SYMPTR
030E E6 08              LDA B 8,X
0310 EE 06              LDX 6,X         GET VALUE
0312 39                 RTS
                * COMPARE TWO STRINGS
                * ON ENTRY [X] = A PARM LIST OF 5 BYTES:
                *       A(STRING1)
                *       A(STRING2)
                *       COUNT OF BYTES TO BE COMPARED
                *
                * ON RETURN IF CC Z IS SET THERE WAS A MATCH
                *
0313 36         COMPAR  PSH A
0314 37                 PSH B
0315 E6 04              LDA B 4,X       GET COUNT
0317 FF 0037 R          STX XSAV        SAVE PARM POINTER
031A FE 0037 R CMP1     LDX XSAV        GET PARM POINTER
031D EE 00              LDX 0,X         GET A(STRING1)
031F A6 00              LDA A 0,X       GET A CHAR
0321 FE 0037 R          LDX XSAV
0324 6C 01              INC 1,X         PTR SET TO NEXT
0326 26 02              BNE CMP2        CHAR IN
0328 6C 00              INC 0,X         STRING1
032A FE 0037 R CMP2     LDX XSAV        GET PARM POINTER
032D EE 02              LDX 2,X         GET A(STRING2)
032F A1 00              CMP A 0,X       COMPARE
0331 26 0C              BNE CDONE       NOT EQUAL
0333 FE 0037 R          LDX XSAV        GET PARM POINTER
0336 6C 03              INC 3,X         PTR SET TO NEXT
0338 26 02              BNE CMP3        CHAR IN
033A 6C 02              INC 2,X         STRING2
033C 5A         CMP3    DEC B           DEC COUNT
033D 26 DB              BNE CMP1        TRY AGAIN
033F 33         CDONE   PUL B           DONE
0340 32                 PUL A
0341 39                 RTS
                * PRINT LOAD MAP
                *
0342 CE 04F9 R PRTSYM   LDX #MAPMSG     "LOAD MAP"
0345 BD 0434 R          JSR PDATA1
0348 BD 045E R          JSR CRLF
034B FE 002C R          LDX SYMTAB
034E BC 002A R          CPX NXTSYM      ANY SYMBOLS?
0351 27 56              BEQ PRTSM3      NO
                *
                *
                * FIND LOWEST VALUED ENTRY TO PRINT
                *
0353 CE FFFF    SORT    LDX #$FFFF
0356 FF 0039 R          STX HIVAL
0359 FE 002C R          LDX SYMTAB
                *
035C FF 0030 R SORT1    STX SYMPTR
035F 86 FF              LDA A #$FF
0361 A1 08              CMP A 8,X       ALREADY PRINTED?
0363 27 1C              BEQ SORT2       YES
                *
0365 08                 INX
0366 08                 INX
0367 08                 INX
0368 08                 INX
0369 08                 INX
036A 08                 INX
036B FF 0032 R          STX STRNG1      POINT TO ENTRY
036E CE 0039 R          LDX #HIVAL
0371 FF 0034 R          STX STRNG2      POINT TO HIVAL
0374 86 02              LDA A #2
0376 B7 0036 R          STA A COUNT
0379 CE 0032 R          LDX #STRNG1     POINT TO PARMS
037C BD 0313 R          JSR COMPAR
037F 25 13              BCS SORT3       ENTRY < HIVAL
                *
0381 FE 0030 R SORT2    LDX SYMPTR
0384 08                 INX
0385 08                 INX
0386 08                 INX
0387 08                 INX
```

```
0388 08                 INX
0389 08                 INX
038A 08                 INX
038B 08                 INX
038C 08                 INX
038D BC 002A R          CPX NXTSYM      END OF TABLE?
0390 27 0F              BEQ SORT4       YES
                *
0392 20 C8              BRA SORT1       NO
                *
0394 FE 0030 R SORT3    LDX SYMPTR      HIVAL:=ENTRY
0397 FF 0027 R          STX DESCRA      SAVE LOWEST ENTRY ADDRESS
039A EE 06              LDX 6,X         GET VALUE
039C FF 0039 R          STX HIVAL
039F 20 E0              BRA SORT2
                *
03A1 CE FFFF    SORT4   LDX #$FFFF
03A4 BC 0039 R          CPX HIVAL       PRINTED ENTIRE LOAD MAP?
03A7 26 01              BNE PRTSM0      NO
                *
03A9 39         PRTSM3  RTS             YES, ALL DONE
                *
                *
03AA FE 0027 R PRTSM0   LDX DESCRA      GET ENTRY TO BE PRINTED
03AD C6 06      PRTSM1  LDA B #6        PRINT 6 CHAR SYMBOL
03AF A6 00      PRTSM2  LDA A 0,X       GET CHAR
03B1 08                 INX
03B2 BD 001E R          JSR OUTEEE
03B5 5A                 DEC B           DONE ?
03B6 26 F7              BNE PRTSM2      NO
                *
03B8 BD 0448 R          JSR OUTS        PRINT A SPACE
                *
03BB BD 0444 R          JSR OUT4HS      PRINT HEX VALUE
03BE E6 00              LDA B 0,X       GET INFO BYTE
03C0 C5 01              BIT B #$01      UNRESOLVED?
03C2 26 06              BNE PRTSM4      NO
                *
03C4 CE 0502 R          LDX #UNRES      YES
03C7 BD 0434 R          JSR PDATA1
                *
03CA C5 80      PRTSM4  BIT B #$80      REDEFINED?
03CC 27 06              BEQ PRTSM5      NO
                *
03CE CE 050E R          LDX #REDEF      YES
03D1 BD 0434 R          JSR PDATA1
                *
                *
03D4 BD 045E R PRTSM5   JSR CRLF
03D7 FE 0027 R          LDX DESCRA      FLAG AS PRINTED
03DA 86 FF              LDA A #$FF
03DC A7 08              STA A 8,X
03DE 7E 0353 R          JMP SORT        GET ANOTHER ENTRY



                * CHECK TO SEE IF MEMORY OVERRUN
                *
03E1 BC 0045 R MEMCHK   CPX UPLIM       OVERRUN?
03E4 27 01              BEQ MEMCKE      YES
                *
03E6 39                 RTS             NO
                *
03E7 BD 045E R MEMCKE   JSR CRLF
03EA CE 04CF R          LDX #MSGD       ERROR MESSAGE
03ED BD 0434 R          JSR PDATA1
03F0 BD 045E R          JSR CRLF
03F3 31                 INS             FIX STACK
03F4 31                 INS
03F5 7E 00F9 R          JMP LOADE       PRINT PARTIAL LOAD MAP
                * I/O ROUTINES
                *
03F8 0002       BXSAV   RMB 2
                *
                * BUILD 16 BIT ADDRESS
                *
03FA 8D 0C      BADDR   BSR INBYTE
03FC B7 03F8 R          STA A BXSAV
03FF 8D 07              BSR INBYTE
0401 B7 03F9 R          STA A BXSAV+1
0404 FE 03F8 R          LDX BXSAV
0407 39                 RTS
                *
                * INPUT A BYTE
                *
0408 8D 09      INBYTE  BSR INHEX
040A 48                 ASL A
040B 48                 ASL A
040C 48                 ASL A
040D 48                 ASL A
040E 16                 TAB
040F 8D 02              BSR INHEX
0411 1B                 ABA
0412 39                 RTS
                *
                * INPUT HEX CHARACTER
                *
0413 BD 001B R INHEX    JSR INEEE
0416 80 30              SUB A #$30
0418 2B 0F              BMI NOTHEX
                *
041A 81 09              CMP A #$09
041C 2F 0A              BLE INHEXR
                *
041E 81 11              CMP A #$11
0420 2B 07              BMI NOTHEX
                *
0422 81 16              CMP A #$16
0424 2E 03              BGT NOTHEX
                *
0426 80 07              SUB A #$7
                *
0428 39         INHEXR  RTS
                *
                * NOT A HEX CHARACTER
                *
0429 86 3F      NOTHEX  LDA A #'?
042B BD 001E R          JSR OUTEEE
042E 20 E3              BRA INHEX
                *
                * PRINT A DATA STRING
                *
0430 BD 001E R PDATA2   JSR OUTEEE
0433 08                 INX
0434 A6 00      PDATA1  LDA A 0,X
0436 81 04              CMP A #4
0438 2A F6              BNE PDATA2
                *
043A 39                 RTS
                *
                * OUTPUT TWO HEX CHARACTERS
                *
043B A6 00      OUT2H   LDA A 0,X
043D 8D 0E      OUT2HA  BSR OUTHL
043F A6 00              LDA A 0,X
0441 08                 INX
0442 20 0D              BRA OUTHR
                *
0444 8D F5      OUT4HS  BSR OUT2H
0446 8D F3      OUT2HS  BSR OUT2H
0448 86 20      OUTS    LDA A #$20
044A 7E 001E R          JMP OUTEEE
                *
044D 44         OUTHL   LSR A
044E 44                 LSR A
044F 44                 LSR A
0450 44                 LSR A
                *
0451 84 0F      OUTHR   AND A #$0F
0453 8B 30              ADD A #$30
0455 81 39              CMP A #$39
0457 23 02              BLS OUTCH
                *
0459 8B 07              ADD A #7
                *
045B 7E 001E R OUTCH    JMP OUTEEE
                *
                *
045E 86 0D      CRLF    LDA A #$0D
0460 BD 001E R          JSR OUTEEE
0463 86 0A              LDA A #$0A
0465 BD 001E R          JSR OUTEEE
                *
0468 39                 RTS
                *
                * MESSAGES
                *
0469 53         SYMFUL  FCC 'SYMBOL TABLE OVERFLOW'
047E 04                 FCB 4
                *
047F 45         MSGL    FCC 'ENTER BASE ADDRESS: '
0493 04                 FCB 4
                *
0494 45         MSGA    FCC 'ENTER UPPER MEMORY LIMIT: '
04AE 04                 FCB 4
                *
04AF 45         MSGB    FCC 'ENTER START OF COMMON: '
04C6 04                 FCB 4
                *
04C7 43         MSGC    FCC 'COMMON:'
04CE 04                 FCB 4
```

```
04CF 2A          MSGD    FCC '****** MEMORY OVERRUN ******'
04EB 04                  FCB 4
                 *
04EC 4C          MSGE    FCC 'LOAD LIMITS:'
04F8 04                  FCB 4
                 *
04F9 4C          MAPMSG  FCC 'LOAD MAP'
0501 04                  FCB 4
                 *
0502 55          UNRES   FCC 'UNRESOLVED '
050D 04                  FCB 4
                 *
050E 52          REDEF   FCC 'REDEFINED'
0517 04                  FCB 4
                 *
0518 53          SAVFIL  FCC 'SAVE LOADED FILE? "Y" OR "N" '
053B 04                  FCB 4
                 * PUNCH: OUTPUT LOAD MODULE IN MIKBUG FORMAT
                 * (BASESV - LAST)
                 *
0536 0001        MCONT   RMB 1
0537 0001        TEMP    RMB 1
                 *
0538 0538      R PUNCH   EQU *
                 *
0538 B6 0044 R PUN11   LDA A LAST+1
053B B0 0042 R         SUB A BASESV+1
053E F6 0043 R         LDA B LAST
0541 F2 0041 R         SBC B BASESV
0544 26 04             BNE PUN22
                 *
0546 81 10             CMP A #16
0548 25 02             BCS PUN23
                 *
054A 86 0F       PUN22 LDA A #15
                 *
054C 8B 04       PUN23 ADD A #4
054E B7 0536 R         STA A MCONT
0551 80 03             SUB A #3
0553 B7 0537 R         STA A TEMP
                 *
0556 CE 05C4 R         LDX #MTAPE1
0559 BD 05B6 R         JSR DDATA1
055C 5F                CLR B
                 *
                 * OUTPUT FRAME COUNT
                 *
055D CE 0536 R         LDX #MCONT
0560 8D 33             BSR PUNT2
                 *
                 * OUTPUT ADDRESS
                 *
0562 CE 0041 R         LDX #BASESV
0565 BD 0595 R         JSR PUNT2
0568 BD 0595 R         JSR PUNT2
                 *
                 * OUTPUT DATA
                 *
056B FE 0041 R         LDX BASESV
                 *
056E BD 0595 R PUN32   JSR PUNT2
0571 7A 0537 R         DEC TEMP
0574 26 F8             BNE PUN32
                 *
0576 FF 0041 R         STX BASESV
0579 53                COM B
057A 37                PSH B
057B 30                TSX
057C BD 0595 R         JSR PUNT2
057F 33                PUL B
0580 FE 0041 R         LDX BASESV
0583 09                DEX
0584 BC 0043 R         CPX LAST
0587 26 AF             BNE PUN11
                 *
0589 CE 05BD R         LDX #EOF
058C BD 05B6 R         JSR DDATA1
                 *
058F BD 0012 R         JSR WREOF
                 *
0592 7E 0018 R         JMP UPDATE
                 *
0595 EB 00       PUNT2   ADD B 0,X
0597 A6 00       OUT2HD  LDA A 0,X
0599 8D 05               BSR OUTHLL
059B A6 00               LDA A 0,X
059D 08                  INX
059E 20 04               BRA OUTHRR
                 *
05A0 44          OUTHLL  LSR A
05A1 44                  LSR A
05A2 44                  LSR A
05A3 44                  LSR A
                 *
05A4 84 0F       OUTHRR  AND A #$F
05A6 8B 30               ADD A #$30
05A8 81 39               CMP A #$39
05AA 23 02               BLS OTHRR
                 *
05AC 8B 07               ADD A #7
                 *
05AE BD 000F R OTHRR   JSR OUTB
05B1 39                RTS
                 *
05B2 BD 000F R DDATA2  JSR OUTB
05B5 08                INX
05B6 A6 00       DDATA1  LDA A 0,X
05B8 81 04               CMP A #4
05BA 26 F6               BNE DDATA2
                 *
05BC 39                RTS
                 *
05BD 0D0A        EOF     FDB $0D0A
05BF 53                  FCC 'S9'
05C1 0D0A                FDB $0D0A
05C3 04                  FCB 4
                 *
05C4 0D0A        MTAPE1  FDB $0D0A
05C6 53                  FCC 'S1'
05C8 04                  FCB 4
                 *
                         END
```

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| BADDR | 03FA | R | MSGC | 04C7 | R | | |
| BASE | 0021 | R | MSGD | 04CF | R | | |
| BASESV | 0041 | R | MSGE | 04EC | R | | |
| BXSAV | 03F8 | R | MSGL | 047F | R | | |
| BYTE | 0024 | R | MTAPE1 | 05C4 | R | | |
| CBAS | 003B | R | NFLAG | 0023 | R | | |
| CBASSV | 003D | R | NOTHEX | 0429 | R | | |
| CDONE | 033F | R | NXTSYM | 002A | R | | |
| CMP1 | 031A | R | OTHRR | 05AE | R | | |
| CMP2 | 032A | R | OUT2H | 043B | R | | |
| CMP3 | 033C | R | OUT2HA | 043D | R | | |
| COMPAR | 0313 | R | OUT2HD | 0597 | R | | |
| COUNT | 0036 | R | OUT2HS | 0446 | R | | |
| CRLF | 045E | RN | OUT4HS | 0444 | R | | |
| DDATA1 | 05B6 | R | OUTB | 000F | RX | | |
| DDATA2 | 05B2 | R | OUTCH | 045B | R | | |
| DESCRA | 0027 | R | OUTEEE | 001E | R | | |
| DESCRC | 0029 | R | OUTHL | 044D | R | | |
| EOF | 05BD | R | OUTHLL | 05A0 | R | | |
| GETB | 000C | RX | OUTHR | 0451 | R | | |
| HICBAS | 003F | R | OUTHRR | 05A4 | R | | |
| HIVAL | 0039 | R | OUTS | 0448 | R | | |
| INBYTE | 0408 | R | PDATA1 | 0434 | RN | | |
| INEEE | 001B | RN | PDATA2 | 0430 | R | | |
| INHEX | 0413 | R | PRTSM0 | 03AA | R | | |
| INHEXR | 0428 | R | PRTSM1 | 03AD | R | | |
| INITIO | 0015 | RX | PRTSM2 | 03AF | R | | |
| LAST | 0043 | R | PRTSM3 | 03A9 | R | | |
| LC | 0025 | R | PRTSM4 | 03CA | R | | |
| LINK | 0000 | RN | PRTSM5 | 03D4 | R | | |
| LKPSY1 | 02DD | R | PRTSYM | 0342 | R | | |
| LKPSY2 | 030B | R | PUN11 | 0538 | R | | |
| LKPSY3 | 0308 | R | PUN22 | 054A | R | | |
| LKPSYM | 02D5 | R | PUN23 | 054C | R | | |
| LOAD | 0047 | R | PUN32 | 056E | R | | |
| LOAD2 | 00A2 | R | PUNCH | 0538 | R | | |
| LOADE | 00F9 | R | PUNT2 | 0595 | R | | |
| LOADE1 | 013B | R | REDEF | 050E | R | | |
| LOADM | 01AA | R | RNIBL | 00E7 | R | | |
| LOADN | 01C5 | R | SAVFIL | 0518 | R | | |
| LOADN3 | 01F1 | R | SORT | 03D3 | R | | |
| LOADN4 | 01F7 | R | SORT1 | 03DC | R | | |
| LOADN5 | 0219 | R | SORT2 | 0381 | R | | |
| LOADN6 | 0204 | R | SORT3 | 0394 | R | | |
| LOADP | 017B | R | SORT4 | 03A1 | R | | |
| LOADP1 | 01A4 | R | STOSY1 | 02A9 | R | | |
| LOADR | 0160 | R | STOSYM | 0295 | R | | |
| LOADX | 0232 | R | STRNG1 | 0032 | R | | |
| LOADX2 | 0255 | R | STRNG2 | 0034 | R | | |
| LOADX3 | 025B | R | SYMEND | 002E | R | | |
| LOADX4 | 0201 | R | SYMFUL | 0469 | R | | |
| LOADX5 | 0277 | R | SYMPTR | 0030 | R | | |
| LOADX6 | 0279 | R | SYMTAB | 002C | R | | |
| MAPMSG | 04F9 | R | TABLES | 0006 | RX | | |
| MCONT | 0536 | R | TEMP | 0537 | R | | |
| MEMCHK | 03E1 | R | UNRES | 0502 | R | | |
| MEMCKE | 03E7 | R | UPDATE | 0018 | RX | | |
| MONITOR | 0009 | RX | UPLIM | 0045 | R | | |
| MSGA | 0494 | R | WREOF | 0012 | RX | | |
| MSGB | 04AF | R | XSAV | 0037 | R | | |

# APPENDIX H

## ASCII Text Listing of the Relocatable Format Object Code for LINK68

The listing on the following page gives the relocatable format object code of the linking loader LINK68 in ASCII text format. This listing can be used to enter the program by hand or to verify the entry of the program via the bar codes given in Appendix I. Note that the ends of lines in this verification listing *do not* represent line feed or carriage return codes within the machine readable text. See *Input Relocatable File Format* on page 15 for a description of the relocation conventions.

Once LINK68 has been bootstraped (see Appendix C), the relocatable file of the linking loader can be run through the loader in order to reposition LINK68 at an arbitrary, more convenient address if low memory is not the ideal location in the user's system. This form of the linking loader object code will not be needed by users who can employ the absolute object code version of LINK68 given in Appendices D or E without further relocation.

Appendix G gives an assembly language source listing for LINK68.

0000P4C494E4B20200000RN8EA0427E0047R7E5441424C4553X7E4D4F4E544F5
2X7E47455442202OX7E4F5554422020X7E5752454F4620X7E494E4954494FX7E
55504441544 5X494E4545452000 1BRN5044415441310434RN43524C462020045
ERN7EE1AC7EE1D1000000000000000000000000000000000000000000000000
0000000000000000000000000000BD0015RFE0007R86A3C602AB01E900B7002FR
F7002EREE00FF002CRFF002AR7F0023RBD045ERCE047FRBD0434RBD03FARBD04
5ERFF0021RFF0041RCE0494RBD0434RBD03FARFF0045RBD045ERCE04AFRBD043
4RBD03FARBD045ERFF003BRFF003DRFF003FRFE0021RBD000CR2552810027F78
15026037E017BR814E26037E01C5R815826037E0232R814D26037E01AAR81522
6037E0160R803081092F0280077D0023R260C48484848B70024R730023R20BBF
60024R1BA700FF0043R08BD03E1R730023R20A9BD045ERBD045ERCE04ECRBD04
34RBD045ERCE0041RBD0444RCE0043RBD0444RBD045ERBD045ERBD0342RBD045
ERCE04C7RBD0434RBD045ERCE003BRBD0444RFE003FRBC003BR270409FF003FR
CE003FRBD0444RBD045ERBD045ERCE0518RBD0434RBD001BR81592606BD045ER
7E0538RBD045ER7E0009R0909A601E600BB0022RF90021RA701E70008BD03E1R
08BD03E1R7E00A2R0909FF0021RA601E600BB003CRF9003BRB7003ERF7003DRB
60040RF6003FRB0003ERF2003DR2406FE003DRFF003FRFE0021R7E00A2R0909A
601E600BB003CRF9003BRA701E70008BD03E1R08BD03E1R7E00A2RC606F70029
R0909A600B70025RA601B70026R090909090909FF0043RFF0027RBD02D5RC1FF
260EBD0295RFE0030R6C08FE0043R7E00A2RC5012709CA80FE0030RE70820EDF
F0027RB60025RF60026RFE0030R6C08A706E707FE0027REE00FF0030RFE0027R
A700E701FE0030R8CFFFF27C4FF0027R20E7C606F70029R090909090909FF002
7RFF0025R0808FF0043RBD02D5RC1FF2612BD0295RFE0025R86FFA700A701FE0
043R7E00A2RFE0030RE608C501270DA606E607FE0025RA700E70120E4EE06FF0
027REE008CFFFF26F6B60025RF60026RFE0027RA700E701FE0025R20C0FE002A
RFF0030RBC002ER2609CE0469RBD0434R7E0009RFE0027RA60008FF0027RFE00
2ARA70008FF002AR7A0029R26E9B60025RA700B60026RA7018600A702080808F
F002AR39FE002CRBC002AR272BFF0030RFF0032RFE0027RFF0034RB60029RB70
036RCE0032RBD0313R2714FE0030R0808080808080808BC002AR26D5C6FF39
FE0030RE608EE06393637E604FF0037RFE0037REE00A600FE0037R6C0126026C
00FE0037REE02A100260CFE0037R6C0326026C025A26DB333239CE04F9RBD043
4RBD045ERFE002CRBC002AR2756CEFFFFFF0039RFE002CRFF0030R86FFA10827
1C080808080808FF0032RCE0039RFF0034R8602B70036RCE0032RBD0313R2513
FE0030R0808080808080808BC002AR270F20C8FE0030RFF0027REE06FF0039
R20E0CEFFFFBC0039R260139FE0027RC606A60008BD001ER5A26F7BD0448RBD0
444RE600C5012606CE0502RBD0434RC5802706CE050ERBD0434RBD045ERFE002
7R86FFA7087E0353RBC0045R270139BD045ERCE04CFRBD0434RBD045ER31317E
00F9R00008D0CB703F8R8D07B703F9RFE03F8R398D0948484848168D021B39BD
001BR80302B0F81092F0A8111 2B0781162E03800739863FBD001ER20E3BD001E
R08A600810426F639A6008D0EA60008200D8DF58DF386207E001ER4444444484
0F8B3081392302 8B077E001ER860DBD001ER860ABD001ER3953594D424F4C205
441424C45204F564552464C4F5704454E5445522042415345204144445245535
33A2004454E5445522055505045522 04D454D4F5259204C494D49543A2004454
E5445522053544 15254204F4620434F4D4D4D4F4E3A2004434F4D4D4D4F4E3A042A2
A2A2A2A2A204D454D4F5259204F56455252554E202A2A2A2A2A2A044C4F41442
04C494D4954533A044C4F4144204D415004554E5245534F4C564544200452454
44546494E4544045341564520 4C4F414445442046494C453F20225922204F522
0224E2220040000B60044RB00042RF60043RF20041R26048110250286 0F8B04B
70536R8003B70537RCE05C4RBD05B6R5FCE0536R8D33CE0041RBD0595RBD0595
RFE0041RBD0595R7A0537R26F8FF0041R533730BD0595R33FE0041R09BC0043R
26AFCE05BDRBD05B6RBD0012R7E0018REB00A6008D05A6000820044444444484
0F8B3081392302 8B07BD000FR39BD000FR08A600810426F6390D0A53390D0A04
0D0A533104

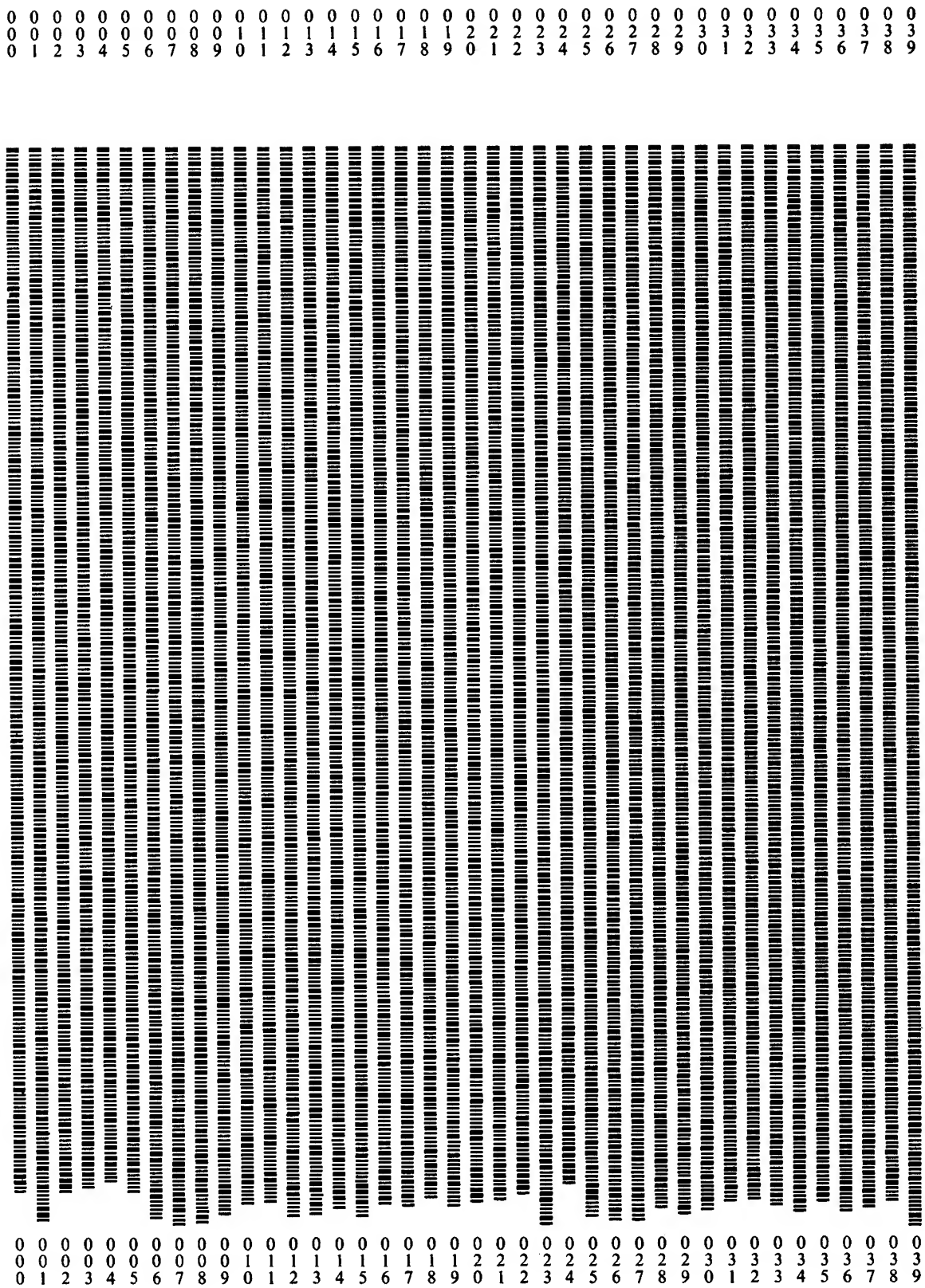
# APPENDIX I

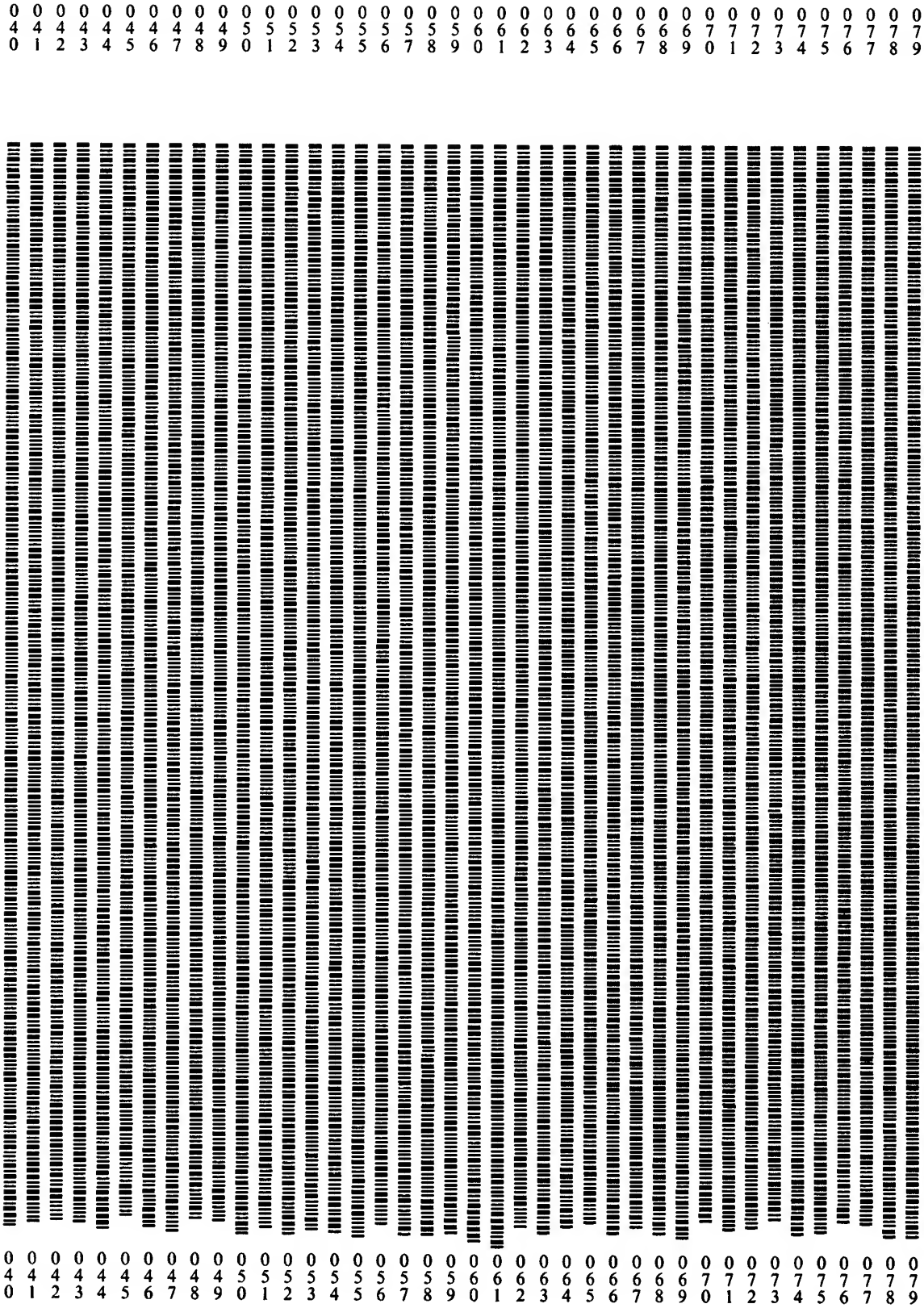
## PAPERBYTE™ Bar Code Representation of Relocatable Format Object Code for LINK68

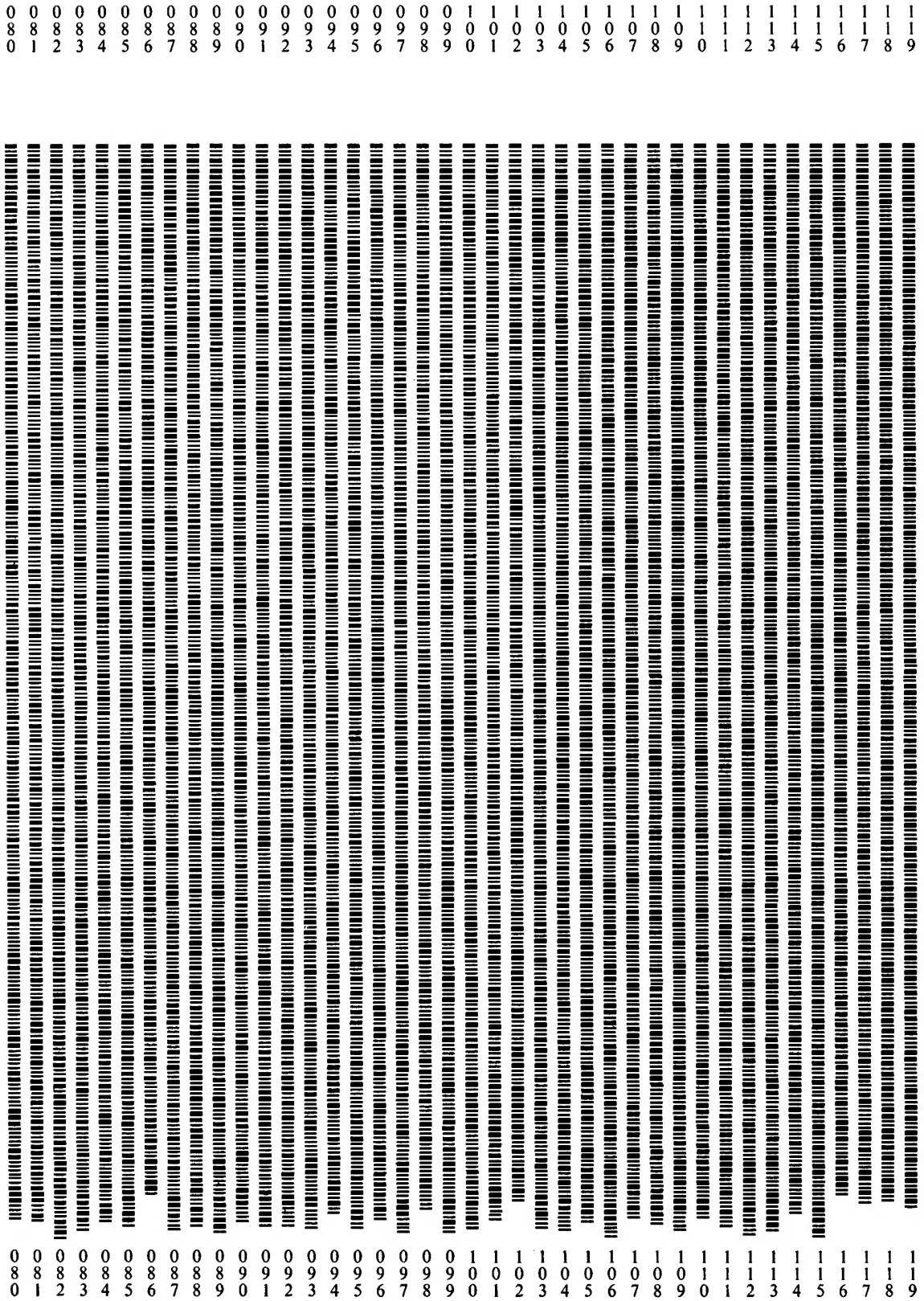
Beginning on the following page is a complete machine readable representation (PAPERBYTE™ bar codes) of the relocatable object code for Grappel and Hemenway's linking loader LINK68. The format is that of an ASCII text string without carriage return or line feed conventions. Appendix H is a direct listing of this file using fixed length lines to make it fit the confines of a printed page. See *Input Relocatable File Format* on page 15 for a description of the relocation conventions.

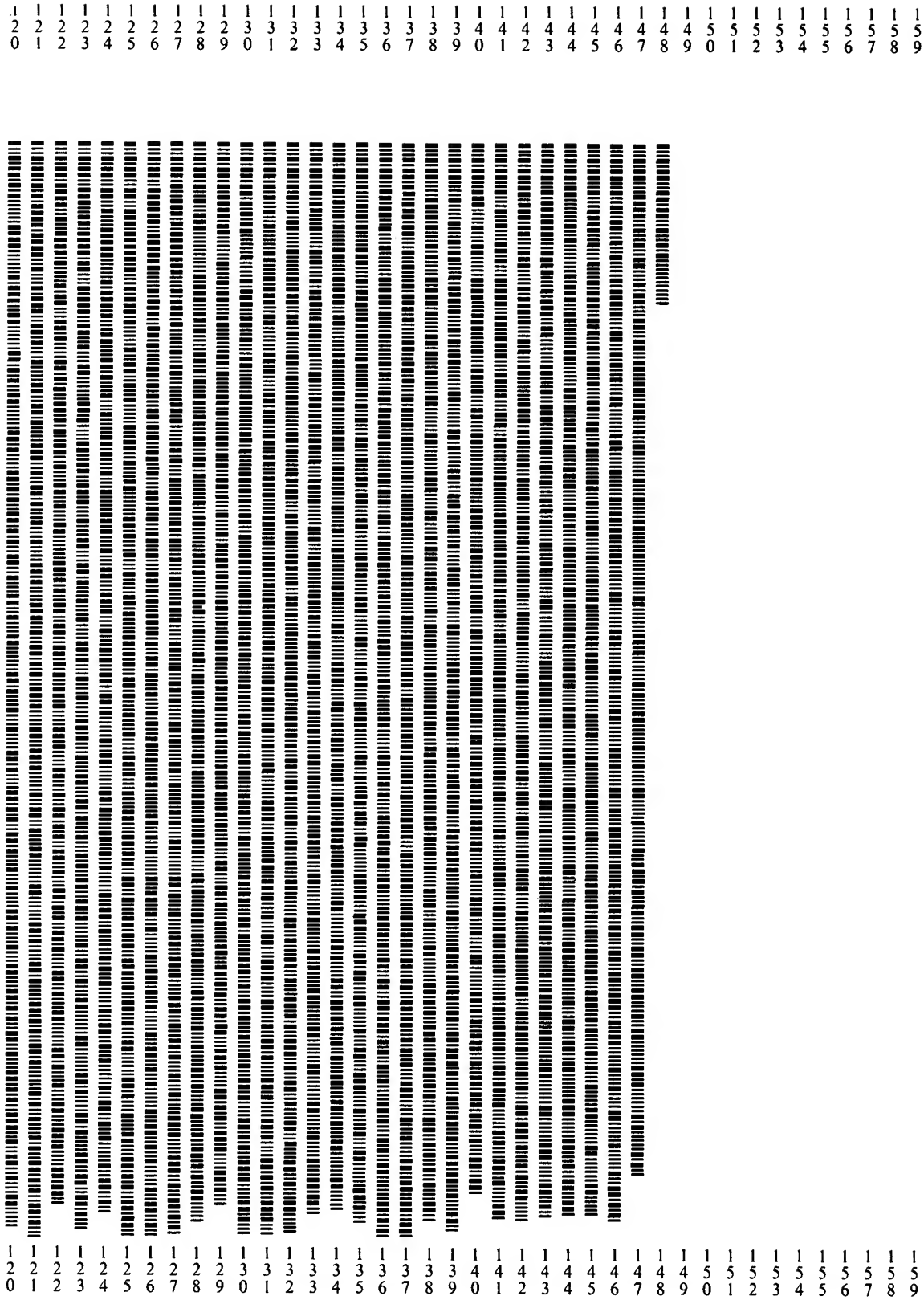
This representation uses the bar code text format, in which each bar code frame (one line of bar codes running from top to bottom of the page) contains a segment of the ASCII relocatable format object text. The text must be loaded into memory and then saved on the user's mass storage device. For details on the text format used in this and other PAPERBYTE™ books, see the PAPERBYTE publication *Bar Code Loader* by Ken Budnick. The book contains a brief history on bar codes, a general bar code loader algorithm with flowcharts, and complete program listings for 6800, 6502, and 8080 and Z-80 based systems.

Once LINK68 has been bootstrapped (see Appendix C), the relocatable file of the linking loader can be run through the loader in order to reposition LINK68 at an arbitrary, more convenient address if low memory is not the ideal location in the user's system. This form of the linking loader object code will not be needed by users who can employ the absolute object code version of LINK68 given in Appendices D or E without further relocation.


000 001 002 003 004 005 006 007 008 009 010 011 012 013 014 015 016 017 018 019 020 021 022 023 024 025 026 027 028 029 030 031 032 033 034 035 036 037 038 039


040 041 042 043 044 045 046 047 048 049 050 051 052 053 054 055 056 057 058 059 060 061 062 063 064 065 066 067 068 069 070 071 072 073 074 075 076 077 078 079


080 081 082 083 084 085 086 087 088 089 090 091 092 093 094 095 096 097 098 099 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119


120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159

# APPENDIX J

## Cassette Tape IO Listing

```
0000 0000    N          NAM TDRIVERS
                        *
                        *       TAPE DRIVERS FOR LINKING LOADER
                        *       C COPYRIGHT 1977 BY
                        *       ROBERT D. GRAPPEL LEXINGTON MASS.
                        *       AND JACK E. HEMENWAY BOSTON MASS.
                        *       ALL RIGHTS RESERVED
                        *
                        *
                        * ROUTINES IN THE LINKING LOADER
                        *
0000 7E 0000 X          EXT PDATAI
0003 7E 0000 X          EXT INEEE
0006 7E 0000 X          EXT CRLF
                        *
                        * ENTRY POINTS IN DRIVER
                        *
0009 01D6    N          ENT TABLES
0009 0009    N          ENT UPDATE
0009 000C    N          ENT MONTOR
0009 0016    N          ENT GETB
0009 0035    N          ENT OUTB
0009 0062    N          ENT WREOF
0009 0051    N          ENT INITIO
                        *
                        * LOCATIONS IN MIKBUG
                        *
0009 7E E0E3 UPDATE     JMP $E0E3
000C 7E E0E3 MONTOR     JMP $E0E3
                        *
000F 0001    CKSUM      RMB 1
0010 0002    INPTR      RMB 2
0012 0002    OTPTR      RMB 2
0014 0002    DXSV       RMB 2
                        *
                        * GET A BYTE RETURN IN A REGISTER
                        *
0016 FF 0014 R GETB     STX DXSV
0019 FE 0010 R          LDX INPTR
001C A6 00              LDA A 0,X       GET A CHAR
001E 81 17              CMP A #$17      ETB ?
0020 26 08              BNE GETB1       NO
                        *
0022 37                 PSH B
0023 BD 0074 R          JSR RDBUF       READ ANOTHER BLOCK
0026 33                 PUL B
0027 24 01              BCC GETB1
                        *
0029 39                 RTS             EOF
                        *
002A A6 00   GETB1      LDA A 0,X       GET CHAR
002C 08                 INX
002D FF 0010 R          STX INPTR
0030 FE 0014 R          LDX DXSV
0033 0C                 CLC
0034 39                 RTS
                        *
                        *
                        * OUTPUT BYTE IN A REGISTER
                        *
0035 FF 0014 R OUTB     STX DXSV
0038 FE 0012 R          LDX OTPTR
003B 8C 05D5 R          CPX #OTBUF+$1FD  FULL?
003E 26 07              BNE OUTB1       NO
                        *
0040 36                 PSH A
0041 37                 PSH B
0042 BD 011E R          JSR WRITBF
0045 32                 PUL A
0046 33                 PUL B
                        *
0047 A7 00   OUTB1      STA A 0,X       SAVE CHAR
0049 08                 INX
004A FF 0012 R          STX OTPTR
004D FE 0014 R          LDX DXSV
0050 39                 RTS
                        *
                        *
0051 CE 01D8 R INITIO   LDX #INBUF
0054 FF 0010 R          STX INPTR
0057 86 17              LDA A #$17
0059 A7 00              STA A 0,X
                        *
005B CE 03D8 R          LDX #OTBUF
005E FF 0012 R          STX OTPTR
0061 39                 RTS
                        *
                        * CLOSE OUTPUT FILE
                        *
0062 BD 011E R WREOF    JSR WRITBF
0065 FE 0012 R          LDX OTPTR
0068 86 04              LDA A #4
006A A7 00              STA A 0,X
```

```
006C 08                 INX
006D FF 0012 R          STX OTPTR
0070 BD 011E R          JSR WRITBF
0073 39                 RTS
                  * READ IN A BLOCK FROM TAPE 1 *
                  *
0074 7F 000F R RDBUF    CLR   CKSUM
0077 CE 01D6 R          LDX   #INBUF     POINT TO INBUF
007A BD 0153 R          JSR   T1INZ      START TAPE 1
007D BD 0176 R RD1      JSR   T1GET      GET CHAR
0080 5D                 TST B            OK ?
0081 26 18              BNE   RD2        NO
                  *
0083 A7 00              STA A 0,X        PUT IN INBUF
0085 08                 INX              BUMP POINTER
0086 81 04              CMP A #$04       EOF?
0088 27 1E              BEQ   RD4        YES
008A 81 17              CMP A #$17       ETB?
008C 26 EF              BNE   RD1        NO
008E 8C 03D7 R          CPX   #INBUF+$1FF OVERRUN ?
0091 27 08              BEQ   RD2        YES
0093 BD 0176 R          JSR   T1GET      GET CKSUM BYTE
0096 7C 000F R          INC   CKSUM      OK ?
0099 27 05              BEQ   RD3        YES
                  *
009B CE 00F7 R RD2      LDX   #TAPERR    BAD
009E 20 0B              BRA   RD5        FINISH UP
                  *
00A0 BD 018E R RD3      JSR   T1ISTP     STOP TAPE 1
00A3 CE 01D8 R          LDX   #INBUF     INIT INPTR
00A6 0C                 CLC
00A7 39                 RTS
                  *
00A8 CE 008E R RD4      LDX   #EOF       EOF MSG
00AB BD 018E R RD5      JSR   T1ISTP     STOP TAPE
00AE BD 0000 R          JSR   PDATA1     PRINT MESSAGE
00B1 BD 0003 R RD6      JSR   INEEE      WAIT FOR "GO"
00B4 81 0D              CMP A #$0D       CR ?
00B6 27 BC              BEQ   RDBUF      TRY AGAIN
                  *
00B8 81 44              CMP A #'D        DONE?
00BA 26 F5              BNE   RD6        NO
00BC 0D                 SEC              YES
00BD 39                 RTS              RETURN
                  *
                  *
00BE 45         EOF     FCC   'EOF:REPOSITION TAPE AND TYPE CR'
00DD 0D0A               FDB   $0D0A      CR,LF
00DF 4F                 FCC   'OR TYPE A "D" IF DONE'
00F4 0D0A               FDB   $0D0A      CR,LF
00F6 04                 FCB   4          EOT
                  *
00F7 54         TAPERR  FCC   'TAPE ERROR:BACK UP A BLOCK & TYPE CR'
0118 0D0A               FDB   $0D0A      CR,LF
011A 04                 FCB   $04        EOT
                  * WRITBF: WRITE OUT OTBUF TO TAPE2
                  *
011B 37         WRITBF PSH B
011F FE 0012 R          LDX   OTPTR
0122 8C 03D8 R          CPX   #OTBUF     EMPTY
0125 27 22              BEQ   WRTBFC     YES
                  *
0127 86 17              LDA A #$17       LOAD ETB
0129 A7 00              STA A 0,X        PUT INTO OTBUF
012B CE 03D8 R          LDX   #OTBUF     POINT TO OTBUF
012E 5F                 CLR B            CLR CKSUM REG
012F BD 0196 R          JSR   T2OTZ      START TAPE
                  *
0132 A6 00      WRTBFA  LDA A 0,X        GET CHAR
0134 EB 00              ADD B 0,X        ADD TO CKSUM
0136 BD 01B1 R          JSR   T2OUT
0139 BC 0012 R          CPX   OTPTR      DONE ?
013C 27 03              BEQ   WRTBFB
                  *
013E 08                 INX              NO
013F 20 F1              BRA   WRTBFA     DO AGAIN
                  *
0141 53         WRTBFB  COM B            FORM CKSUM
0142 17                 TBA              BYTE
0143 BD 01B1 R          JSR   T2OUT
0146 BD 01BE R          JSR   T2OSTP     STOP TAPE
                  *
0149 CE 03D8 R WRTBFC   LDX   #OTBUF
014C FF 0012 R          STX   OTPTR      INIT OTPTR
014F 33                 PUL   B
0150 39                 RTS
                  * TAPE DRIVERS:
                  *
                  *
0151 8010       TP1ST   EQU $8010
0151 8011       TP1DAT  EQU $8011
0151 8014       TP2ST   EQU $8014
```

```
0151 8015       TP2DAT EQU $8015
0151 0002       TXSV   RMB 2
                *
                *
                * START TAPE FOR A READ:
                *
0153 FF 0151 R  T1INZ  STX TXSV
0156 36                PSH A
0157 86 17             LDA A #$17    MASTER RESET, RTS:=0
0159 B7 8010           STA A TP1ST
                *
015C 86 5D             LDA A #$5D    RTS:=1
015E B7 8010           STA A TP1ST
                *
0161 CE 0280           LDX #$0280    DELAY 1 SEC
0164 BD 01CE R         JSR TDELY
                *
0167 86 57             LDA A #$57    MASTER RESET
0169 B7 8010           STA A TP1ST
016C 86 5D             LDA A #$5D    RTS:=1
016E B7 8010           STA A TP1ST
0171 32                PUL A
0172 FE 0151 R         LDX TXSV
0175 39                RTS
                *
                * READ A BYTE
                *
0176 F6 8010   T1GET   LDA B TP1ST   GET STATUS
0179 C5 01             BIT B #$01    RDRF?
017B 27 F9             BEQ *-5       NO
                *
017D C5 70             BIT B #$70    ERRORS?
017F 27 01             BEQ *+3       NO
                *
0181 39                RTS           YES
                *
0182 B6 8011           LDA A TP1DAT GET BYTE
0185 16                TAB
0186 FB 000F R         ADD B CKSUM   FORM CHECKSUM
0189 F7 000F R         STA B CKSUM
018C 5F                CLR B
018D 39                RTS
                *
                * STOP TAPE AFTER A READ
                *
018E 36        T1ISTP  PSH A
018F 86 17             LDA A #$17
0191 B7 8010           STA A TP1ST
0194 32                PUL A
0195 39                RTS
                *
                * START TAPE FOR OUTPUT
                *
0196 37        T2OTZ   PSH B
0197 36                PSH A
0198 FF 0151 R         STX TXSV
019B C6 17             LDA B #$17    MASTER RESET
019D F7 8014           STA B TP2ST
01A0 C6 5D             LDA B #$5D    RTS:=1
01A2 F7 8014           STA B TP2ST
                *
01A5 CE 0500           LDX #$0500    DELAY 2 SECS.
01A8 BD 01CE R         JSR TDELY
                *
01AB 32                PUL A
01AC 33                PUL B
01AD FE 0151 R         LDX TXSV
01B0 39                RTS
                *
                * WRITE A BYTE TO TAPE
                *
01B1 37        T2OUT   PSH B
01B2 F6 8014   T2OUTA  LDA B TP2ST    GET STATUS
01B5 C5 02             BIT B #$02     READY?
01B7 27 F9             BEQ T2OUTA     NO
                *
01B9 B7 8015           STA A TP2DAT   YES, WRITE BYTE
01BC 33                PUL B
01BD 39                RTS
                *
                * STOP TAPE AFTER A WRITE
                *
01BE 4F        T2OSTP  CLR A          WRITE PAD CHARS
01BF BD 01B1 R         JSR T2OUT
01C2 BD 01B1 R         JSR T2OUT
01C5 BD 01B1 R         JSR T2OUT
01C8 86 17             LDA A #$17
01CA B7 8014           STA A TP2ST
01CD 39                RTS
                *
                *
01CE 4F        TDELY   CLR A
01CF 4C        TDELY1  INC A
```

```
01D0 26 FD            BNE TDELY1
                      *
01D2 09               DEX
01D3 26 FA            BNE TDELY1
01D5 39               RTS
                      *
                      *
                      *
01D6 05D9     R TABLES FDB *+$0403
01D8 01D8     R INBUF  EQU *
01D8 03D8     R OTBUF  EQU *+$200
                      *
                      END
```

```
CKSUM   000F R
CRLF    0006 RX
DXSV    0014 R
EOF     00BE R
GETB    0016 RN
GETB1   002A R
INBUF   01D8 R
INEEE   0003 RX
INITIO  0051 RN
INPTR   0010 R
MONTOR  000C RN
OTBUF   03D8 R
OTPTR   0012 R
OUTB    0035 RN
OUTB1   0047 R
PDATA1  0000 RX
RD1     007D R
RD2     009B R
RD3     00A0 R
RD4     00A8 R
RD5     00AB R
RD6     00B1 R
RDBUF   0074 R
T1GET   0176 R
T1INZ   0153 R
T1ISTP  018E R
T2OSTP  01BE R
T2OTZ   0196 R
T2OUT   01B1 R
T2OUTA  01B2 R
TABLES  01D6 RN
TAPERR  00F7 R
TDELY   01CE R
TDELY1  01CF R
TDRIVE  0000 RN
TP1DAT  8011
TP1ST   8010
TP2DAT  8015
TP2ST   8014
TXSV    0151 R
UPDATE  0009 RN
WREOF   0062 RN
WRITBF  011E R
WRTBFA  0132 R
WRTBFB  0141 R
WRTBFC  0149 R
```

# APPENDIX K

## ICOM Floppy Disk IO Listing

```
0000 0000    N          NAM DDRV
                        *
                        *       DISK DRIVERS FOR LINKING LOADER
                        *       C COPYRIGHT 1977 BY
                        *       ROBERT D. GRAPPEL LEXINGTON MASS.
                        *       AND JACK E. HEMENWAY BOSTON MASS.
                        *       ALL RIGHTS RESERVED
                        *
                        *
                        * ENTRY POINTS IN DRIVER
                        *
0000 002C    N          ENT TABLES
0000 0003    N          ENT UPDATE
0000 0006    N          ENT MONTOR
0000 000B    N          ENT GETB
0000 0017    N          ENT OUTB
0000 0023    N          ENT WREOF
0000 002B    N          ENT INITIO
0000 0000    N          ENT RESTR
                        *
                        * LOCATIONS IN PROM BOOTSTRAP FDOS
                        *
0000 7E E838   RESTR     JMP $E838
0003 7E E820   UPDATE    JMP $E820
0006 7E E0E3   MONTOR    JMP $E0E3
0009 000D      OCNTR     EQU $000D
0009 E929      RIX       EQU $E929
0009 E9AA      WRT       EQU $E9AA
0009 0002      DXSV      RMB 2
                        *
                        *
                        * GET A BYTE RETURN IN A REGISTER
                        * CARRY FLAG SET IF EOF
                        *
000B 37        GETB      PSH B
000C FF 0009 R           STX DXSV
000F BD E929             JSR RIX
0012 FE 0009 R           LDX DXSV
0015 33                  PUL B
0016 39                  RTS
                        *
                        * OUTPUT BYTE IN A REGISTER
                        *
0017 37        OUTB      PSH B
0018 FF 0009 R           STX DXSV
001B BD E9AA             JSR WRT
001E FE 0009 R           LDX DXSV
0021 33                  PUL B
0022 39                  RTS
                        *
                        * WRITE NULLS TO LAST SECTOR
                        *
0023 4F        WREOF     CLR A
0024 BD E9AA             JSR WRT
0027 91 0D               CMP A OCNTR
0029 26 F8               BNE WREOF
                        *
002B 39        INITIO RTS   DUMMY INIT
                        *
                        * START OF LINKING LOADER TABLES
0061                    *
0062   002C 002E    R TABLES FDB *+2
0063                    *
0064                          END
```

```
DDRV    0000 RN
DXSV    0009 R
GETB    000B RN
INITIO  002B RN
MONTOR  0006 RN
OCNTR   000D
OUTB    0017 RN
RESTR   0000 RN
RIX     E929
TABLES  002C RN
UPDATE  0003 RN
WREOF   0023 RN
WRT     E9AA
```

# Index



**Note:** The page numbers in **bold** type face indicate either the definition or the primary reference to the item.

BYTE Publications, Inc.
Production Credits

Blaise Liffick – Technical Editor
Edmond Kelly Jr. – Production Manager
Patricia Curran – Production Assistant
Walter Banks, University of Waterloo, CCNG, Bar Codes

George Banta Company, Printing
Dawson Advertising, Cover Art

## A Note About Bar Codes . . .

Bar codes are the newest form of machine readable data representation. They are used in all PAPERBYTE™ software products in BYTE magazine articles and self contained book publications and combine efficiency of space, low cost, and ease of data entry with the need for mass produced machine readable representations of software. Bar codes were originally used for product identification in inventory control and supermarket checkout applications. Today, because of their direct binary representation of data, they are an ideal computer compatible communications medium. In the application of bar codes to software distribution (such as PAPERBYTE books and articles), the use of a simple but reliable optical scanning wand and an appropriate program provides a convenient means for the user to acquire software.

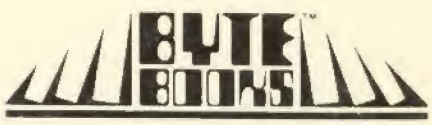
Our intent in making PAPERBYTE software available in bar code form is to provide a method of conveying machine readable information from documentation to the memories and mass storage of a user's system on a one time basis. We suggest that the user of software obtained in this manner should locally record the data on the mass storage devices of his system after the data has been scanned from the printed page. The PAPERBYTE bar code representations provide a standardized means of obtaining the data, but they cannot be compared to the convenience of local mass storage devices such as floppy disks, digital cassettes or audio cassettes. Thus if repeated use of the software obtained from bar code is anticipated, we recommend that the user make a copy on some form of magnetic medium.

*Bar Code Loader* by Ken Budnik, the first in the PAPERBYTE series of software books, provides a brief history of bar codes, a look at the PAPERBYTE bar code format including flowcharts, a general bar code loader algorithm and well documented programs with complete implementation and checkout procedures for 6800, 6502 and 8080/Z-80 based systems.

# LINK68

is a one pass linking loader which allows separately translated relocatable object modules to be loaded and linked together to form a single executable load module. It produces a Load Map and a load module in Motorola MIKBUG loader format. The Linking Loader requires 2 K bytes of memory, a system console such as a Teletype, a system monitor such as the Motorola MIKBUG read only memory program or the ICOM Floppy Disk Operating System (FDOS), and some form of mass file storage such as dual cassette recorders or a floppy disk.

It was the express purpose of the authors of this book to provide everything necessary so that the user can easily learn what he or she needs to know about the system. By providing not only the source code and PAPERBYTE™ bar code listings, but also a detailed description of the major routines of the Linking Loader, they intend to provide the user with an opportunity to learn about the nature of linking loader design and implementation, as well as simply acquiring a useful software tool.




ISBN 0-931718-09-0